View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Trepo - Institutional Repository of Tampere University

TAMPERE UNIVERSITY OF TECHNOLOGY

JANNE HUSBERG

DISTRIBUTED REPAIR OF DIGITAL BROADCASTS

Master of Science Thesis

Examiner: Prof. Jarmo Harju
Examiner and topic approved by the
Faculty of Computing and Electrical
Engineering Council on 15.08.2012

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
HUSBERG, JANNE: Distributed Repair of Digital Broadcasts
Master of Science Thesis, 40 pages, 6 Appendix pages
October 2012
Major: Communication Networks and Protocols
Examiners: Professor Jarmo Harju, D.Sc. (Tech.) Jani Peltotalo
Keywords: Peer-to-peer; stream repair; DVB; data differencing; hashing

Digital broadcasts are a highly efficient way of transmitting data to a large number of receivers, but they are typically not designed for reliable transmission. In this thesis we attempt to design a fully peer-to-peer repair system that can operate on digital broadcasts that have no inherent support for reliability. We exchange hashed representations of the broadcasted stream between peers and use data differencing techniques along with majority voting to detect errors in the transmission with a high probability of success.

We find that the theory behind the system is sound, but that transmitters in large broadcasting networks may slightly modify the transmitted data in such a way that extensive canonicalization is required for data differencing techniques to function properly. The presented system offers increased robustness compared to existing systems, but is significantly more complex.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
HUSBERG, JANNE: Distributed Repair of Digital Broadcasts
Diplomityö, 40 sivua, 6 liitesivua
Lokakuu 2012
Pääaine: Tietoliikenneverkot ja protokollat
Tarkastajat: Professori Jarmo Harju, TkT Jani Peltotalo
Avainsanat: Vertaisverkko; tietovuon korjaus; DVB; tiedon erotus; viestitiiviste

Digitaaliset yleislähetykset välittävät tehokkaasti tietoa suurelle määrälle vastaanottajia, mutta näitä lähetyksiä ei yleensä ole suunniteltu luotettavaan tiedonvälitykseen. Tässä diplomityössä on suunniteltu vertaisverkon ja tavanomaisen laajakaistaisen Internet-yhteyden hyväksikäyttöön perustuva virheenkorjauspalvelu, joka toimii apuna yleislähetysverkossa, jossa ei ole luotettavuutta tukevia ominaisuuksia. Menetelmässä vaihdetaan tietovuon tiivisteitä vertaisverkon jäsenten kesken ja niihin sovelletaan erotusalgoritmeja. Enemmistöäänestyksen avulla tunnistetaan virheet suurella todennäköisyydellä.

Tutkimuksen tuloksena voidaan todeta, että järjestelmän teoria on pätevä, mutta järjestelmä vaatii suuren määrän tiedon yhdenmukaistamista toimiakseen suurissa lähetysverkoissa, koska on mahdollista, että jotkut lähettimet muuntavat tietovuon kehystystä. Kuvattu järjestelmä tarjoaa olemassa oleviin järjestelmiin verrattuna parempaa robustisuutta, mutta vaatii huomattavasti enemmän laskentatehoa.

PREFACE

This thesis was written as a self-funded study.

I would like to thank the examiners of this thesis: Professor Jarmo Harju and D.Sc. (Tech.) Jani Peltotalo for their guidance and feedback.

I would also like to thank my family and friends for their continuous and unquestioning support in my many years of study.

Tampere, September 2012

Janne Husberg

CONTENTS

TERMS AND SYMBOLS

ACK	Acknowledgement
ARQ	Automatic Repeat reQuest
APSK	Amplitude and Phase-Shift Keying
AU	Access Unit
CA	Conditional Access
COFDM	Coded Orthogonal Frequence-Division Multiplexing
CRC32	32-bit Cyclic Redundancy Check. A hash function.
CRR	Compound Receiver Report
CSP	Content Service Provider
CSR	Compound Sender Report
DHT	Distributed Hash Table
DNA	Deoxyribonucleic acid
DTS	Decoding Time Stamp
DVB	Digital Video Broadcasting
DVB-C	Digital Video Broadcasting - Cable
DVB-S	Digital Video Broadcasting - Satellite
DVB-T	Digital Video Broadcasting - Terrestrial
DVR	Digital Video Recorder
ECC	Error-Correcting Codes
EIT	Event Information Table
EPG	Electronic Program Guide
ES	Elementary Stream
FF	RTCP feed-forward message
FB	RTCP feedback message
FEC	Forward Error Correction
FDM	Frequence Division Multiplexing
hash function	A function that maps a large data set to a smaller data set.
hash sum	The output of a hash function.
IP	Internet Protocol
LAN	Local Area Network

LCS	Longest Common Subsequence
MPEG	Moving Picture Experts Group
mux	Multiplex
NAK	Negative acknowledgement
P2P	Peer-to-Peer
PCI	Peripheral Component Interconnect. An internal hardware interface for desktops.
PCR	Program Clock Reference
PID	Packet Identifier
Pearson hash	An 8-bit hash function designed by Peter K. Pearson
PES	Packetized Elementary Stream
PRBS	Pseudo Random Binary Sequence
PS	Program Stream
PSI	Program Specific Information
PTS	Presentation Time Stamp
PUSI	Program Unit Start Indicator
QAM	Quadrature Amplitude Modulation
QEF	Quasi-Error Free
QoS	Quality of Service
QPSK	Quadrature Phase-Shift Keying
RET	Label for DVB-IPTV retransmission servers
RS	Reed-Solomon
RTCP	RTP Control Protocol
RTT	Round-Trip Time
RTP	Real-time Transport Protocol
SES	Shortest Edit Script
SNR	Signal-to-Noise Ratio
SPoF	Single Point of Failure
TDM	Time Division Multiplexing
TEI	Transport Error Indicator
TS	Transport Stream

UDP User Datagram Protocol

USB Universal Serial Bus. An external hardware interface for all types of computers.

VoD Video on Demand

1. INTRODUCTION

Throughout the day, data is being broadcasted on multiple mediums, both wired and wireless. The data may have only a single intended recipient or supply millions of consumers with a wealth of information.

Thought mankind has attempted to account for the nature of the mediums that the signal travels through, the successful reception of that signal at the destination is still a matter of chance. Interference may be an effect of other electronics' operation, matter blocking the progress of the signal, regional or even solar weather.

We therefore classify purely unidirectional transmissions as being unreliable. We may encode information in the signal in ways that make it more robust to transient transmission errors at the cost of throughput, but there will always be a probability that the signal is corrupted or blocked in its travels.

The Two-Army Problem[1] reminds us that not even a bidirectional communication channel is safe from the fickle whims of Lady Luck when reception of any one transmission is uncertain, but with bidirectional communication the chances are in our favour in such a way that we can effectively call the communication reliable. As such, bidirectional communication channels are now the preferred way of transmitting most kinds of information.

The last bastion of unidirectional transmissions is broadcasted audiovisual data. Radio and television have had their flings with interactive services run over side-channels and Internet Protocol (IP)-based transmissions are slowly eroding their broadcast foundations with the rise of broadband Internet, but large networks of purely unidirectional transmitters still function all over the globe. These networks are the most cost-efficient medium for transmitting the highly bandwidth-hungry data to such a large number of consumers. The data is simultaneously delivered to all consumers within the transmitter's range at a fixed bandwidth-cost.

The networks have been operational for decades, with a switch from analog to

digital techniques having taken place around the turn of the millenium, but they still operate in a mostly identical fashion. As such, changing the infrastructure and standards on a national, or even global, scale is a slow and costly procedure. This leaves us with an efficient and large-scale unidirectional channel, but with no simple way to guarantee reliable delivery of the transmission's content.

A large part of the consumers that receive the signal are more likely than not to have full Internet connectivity. This side-channel could provide reliable signalling and data repair, but we still require method of detecting and correcting errors in the unidirectional transmission.

Audiovisual data is unique in being both information-heavy and robust to information loss. This is primarily thanks to the human brain, which can compensate for errors as long as it continuously receives a fluid stream of continuous data. This allows us to transmit audiovisual data over unreliable channels as long as the number of errors is kept relatively low.

Data sent over channels with repair capabilities are generally pre-framed as discrete and easily verifiable packages, but audiovisual data is sent as a stream of random access data with minimal amounts of framing. This presents a problem for us, as the methods used for guaranteeing reliable delivery expect certain framing to perform efficiently. The existing infrastructure used to send audiovisual data cannot be easily modified without modifications to every receiver of the data.

One advantage that we have in radio and television broadcast networks is their large scale. Each transmission is being broadcasted over a large area and, with the ubiquity of bidirectional connectivity between these receivers, this gives us access to a large network of receivers. By correlating each copy of the signal to the others and examining any differences, we may detect and possibly correct any errors in the signal.

In this thesis we will attempt to utilize data differencing techniques to perform fully Peer-to-Peer (P2P) error detection and correction on data received from an unidirectional network that has no inherent support for it. We will focus on the audiovisual data of Digital Video Broadcasting (DVB) networks, but we theorize that the same methods could be useful for any network where multiple receivers of

a common transmission have bidirectional connectivity to each other. The design does not require, nor expect, bidirectional connectivity to the producer of the unidirectional transmission, as that would add a barrier to independent deployment and prevent its function in the worst case scenarios where the system would provide the most benefit compared to existing solutions.

Our primary motivation for this thesis is enabling peer-to-peer repair systems to function in existing DVB networks. Transmission errors in the networks are largely regional and can be assumed to be independent and identically distributed (i.i.d.) random events on a large scale. The probability of an error occuring for all independently received transmissions approaches zero as the number of receivers increases. This should allow a peer-to-peer system to provide near-perfect error correction without carrier assistance or network modifications.

We begin by exploring the DVB standard and any existing systems that could perform error correction in Chapter 2, continue with the basic theory and a rough design of our proposed system in Chapter 3 and finally attempt to assess the solution in Chapter 4.

2. DIGITAL VIDEO BROADCASTING

DVB is one of the most widely used unidirectional transmission system available to consumers in the eastern hemisphere. A large part of the DVB platform's protocols also occur in the rest of the world. National networks generally provide coverage to the majority of its population and as such provide an excellent platform for distributed systems research. Each receiver has a subset of the transmitted stream in their buffer that is determined by the receiver's configuration and the transmission delay between the receiver and a transmitter. Each receiver's buffer may contain a number of minor and major modifications to their content due to transmission errors and regional broadcasting, but for the most part the content is identical. By correlating the contents of each peer's buffer, it may be possible to detect the differences and transform each peer's buffer into a canonical form, free of any transmission errors or other unwanted content.

DVB networks come in multiple variants and may be split into cable, terrestrial and satellite networks. The networks may each carry unique content or share content between different network variants. The networks consist of multiple transmitters, either connected together by a wired backbone or feeding off another network. A transmitter's range may be anywhere from a single building, as is the case for cable networks in some appartment buildings, to multiple nations, when dealing with a satellite transmitter. National networks may therefore consist of dozens to thousands of transmitters, which may all transmit a slightly modified signal. This can be seen in Figure 2.1, the coverage map of Digita's terrestrial DVB network.

The DVB platform itself is primarily designed for unidirectional use, but umbiquitous Local Area Network (LAN) and Internet connectivity provides each receiver with a bidirectional link to a multitude of other receivers. The protocols used by DVB do not rely on data retransmission for error correction, as there is no mandatory return channel, but instead use coding to detect and correct any errors at the

Figure 2.1: Approximate DVB-T coverage of Digita's network. Each dot surrounded by a semi-circular coverage area is a transmitter. Adapted from [2] and [3]

receiver's end. This presents us with a problem, as the lack of packet framing that enables traditional retransmission makes the implementation of an auxiliary repair service difficult.

In the following sections we will examine the protocol stack and the link layer that provides error correction. We will also take a look at a few existing solutions that provide retransmission to DVB networks.

2.1 PROTOCOL STACK

The different protocols used in the DVB platform[4][5][6][7] can be visualized as a four-layer stack. A coding layer, defining the various encodings produced by audio and video encoders, a synchronization layer, defining a generic protocol for synchronizing various substreams, and a multiplexing layer, which defines protocols for combining substreams into a single bitstream. A link layer encodes the resulting bitstream for the physical medium. The function of the layers is illustrated in Figure 2.2.

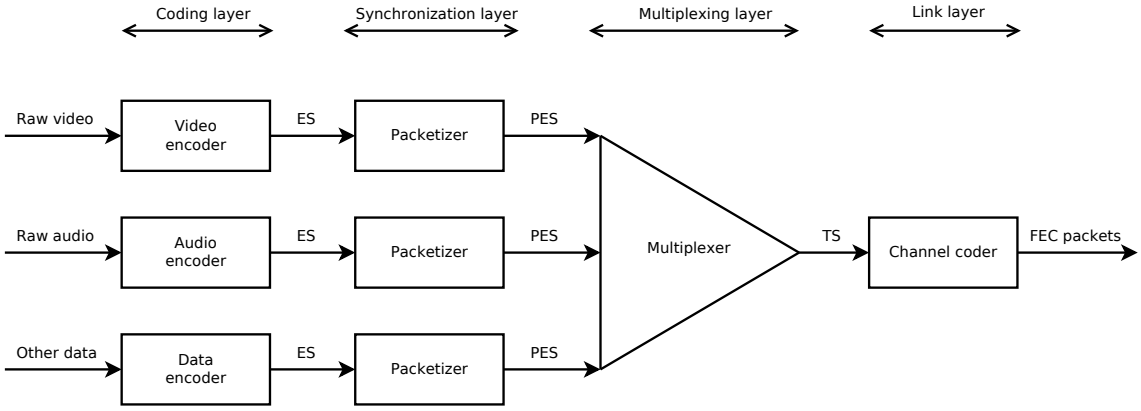

Figure 2.2: The various layers of the DVB stack.

2.1.1 CODING LAYER

The coding layer accepts data from encoders, which generally consists of compressed audiovisual content and attached auxiliary data, but which may also carry generic data. The layer produces discrete Access Units (AU), which are sent to the synchronization layer as an Elementary Stream (ES). Each elementary stream carries only a single video, audio or other bitstream. The access units may still be distinguished

as discrete elements by a lower layer, but do not necessarily encode their position in the stream outside of their implicit order in the sequence.

2.1.2 SYNCHRONIZATION LAYER

The access units are fitted into Packetized Elementary Stream (PES) packets of varying length. The PES header contains a 16-bit length field for its payload, but that length may be set to zero for unbounded video. The PES encapsulation provides substream synchronization for video and audio streams with a common time line by injecting Decoding and Presentation Time Stamps (DTS and PTS) into the packets' header. The separate DTS is needed to ensure that all data required to decode a video picture or audio sample is in the decoding buffer, as some elements may require both preceding and following elements be available before decoding. The PES packet structure is illustrated in Figure 2.3.

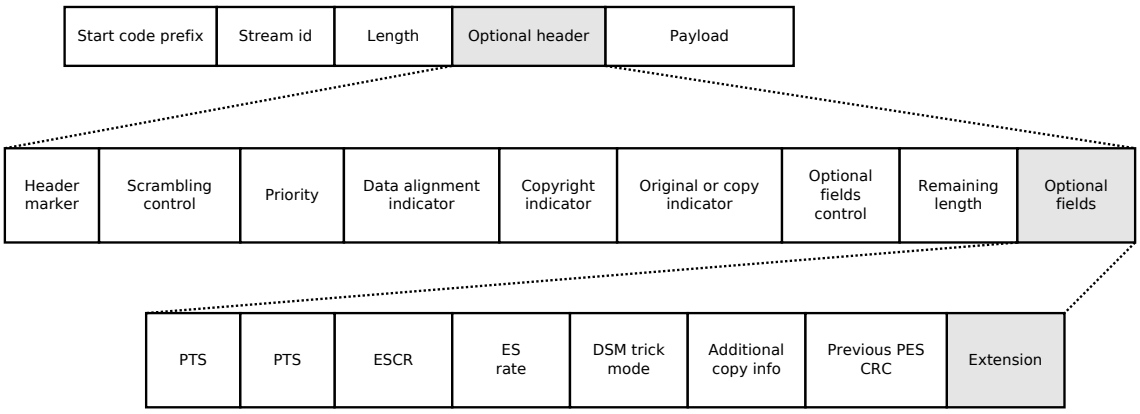

Figure 2.3: Packetized Elementary Stream packet structure.

2.1.3 MULTIPLEXING LAYER

The PES packets are further encapsulated in Transport Stream (TS) packets. These are fixed-size 188-byte packets that provide multiplexing, error detection and additional synchronization.

The packet has a 4-byte header, with an optional variable-length adaptation field, followed by up to 184-bytes of PES data bytes. A transport stream may consist of one or more PES multiplexed together using individual TS packets' Packet IDentifier (PID) header field. The PES may be substreams of one or more programmes with

independent time lines. The individual substreams can be associated to a common programme using Program-Specific Information (PSI) tables multiplexed into the TS.

A TS packet contains a Transport Error Indicator (TEI) bit that allows the underlying link layer to signal an uncorrectable error in the packet and a 4-bit Continuity Counter that is incremented each time a payload is present. These may be used to detect the most common transmission errors.

A Program Clock Reference (PCR) field can be present in the adaptation field that allows the demultiplexer and multiplexer to recover the clock from network jitter. The DVB standard requires that the PCR be present at least every 100 ms[7].

A Payload Unit Start Indicator (PUSI) is set whenever a new PES packet starts at the first byte of the TS packet's payload. A PES may not start anywhere but at the first byte of the payload, so the minimum efficient length of a PES is the length of the payload. While padding may be used to carry shorter PES packets, this would be inefficient and, as such, rarely happens.

Most header fields carry data specific to the indicated PID, so demultiplexing the streams can largely be accomplished by switching on the PID field. The TS packet structure is illustrated in Figure 2.4.

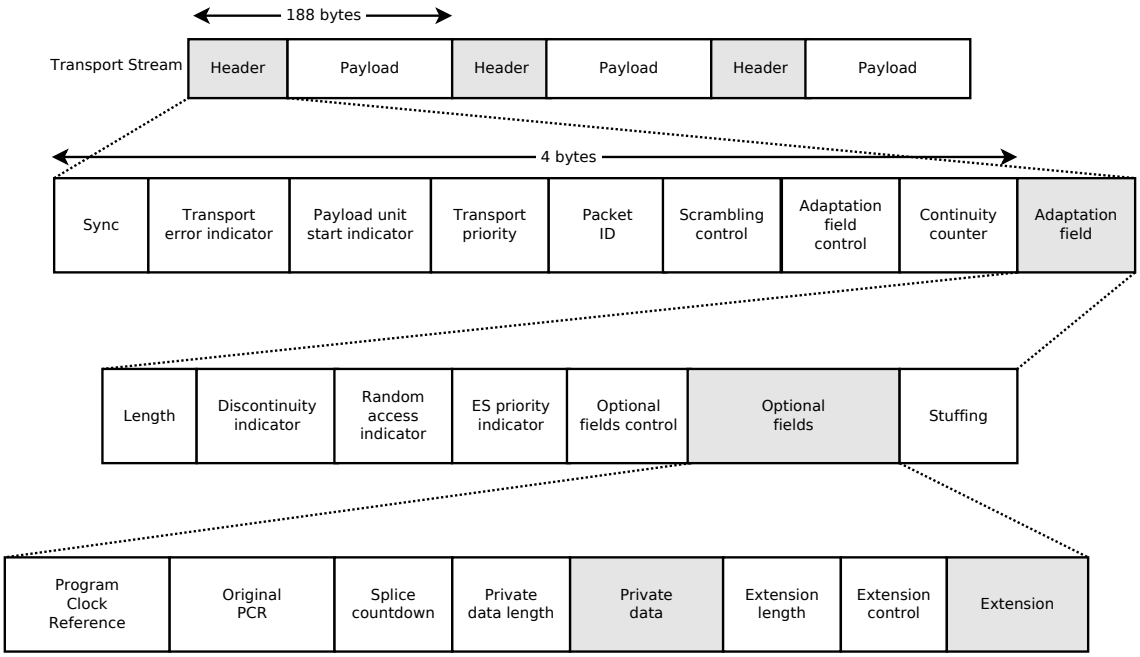

Figure 2.4: Transport Stream packet structure.

Both TS packet headers and PES packet headers contain scrambling control fields. These may be used in conjunction with auxiliary encryption systems to provide Conditional Access (CA) to the content. The standard allows scrambling to take place at either level.

2.1.4 LINK LAYER

A transport stream consisting of multiple independent programming streams can be referred to as a multiplex or "mux" of TV channels. A mux typically carries a fixed number of streams with a total bandwidth of between 20-60 Mbps, determined by the capacity of the DVB channel. The bandwidth of a DVB channel depends on the physical media it is transmitted over, the alloted frequency bands, and coding and modulation techniques.

The main DVB varieties are DVB – Satellite (DVB-S), DVB – Terrestrial (DVB-T) and DVB – Cable (DVB-C). The varieties transmit digital data using Coded Orthogonal Frequency-Division Multiplexing (COFDM), Time Division Multiplexing (TDM) or Frequency Division Multiplexing (FDM) with various levels of Quadrature Amplitude Modulation (QAM), Quadrature Phase-Shift Keying (QPSK) and Amplitude and Phase-Shift Keying (APSK).

All three varieties apply Forward Error Correction (FEC) codes to the TS packets before modulation to combat transmission errors. FEC adds redundancy to the data in such a way that the reception of only a subset of the bits still allows reconstruction of the original data. The Error-Correcting Codes (ECC) provide both error detection and error correction. FEC reconstruction is illustrated in Figure 2.5.

The transport stream is first sent through an energy dispersal stage where a Pseudo Random Binary Sequence (PRBS) generator with a period of eight TS packets randomizes the data to ensure adequate binary transitions. Each randomized packet then has a Reed-Solomon RS(204, 188, T=8) shortened code applied to it. The applied code provides error correction for up to 8 errors per packet.

The resulting 204 byte packets are run through a convolutional interleaver with a depth of I=12. By interleaving the data, burst error protection is increased by spreading the errors over multiple RS-protected packets. Punctured convolutional

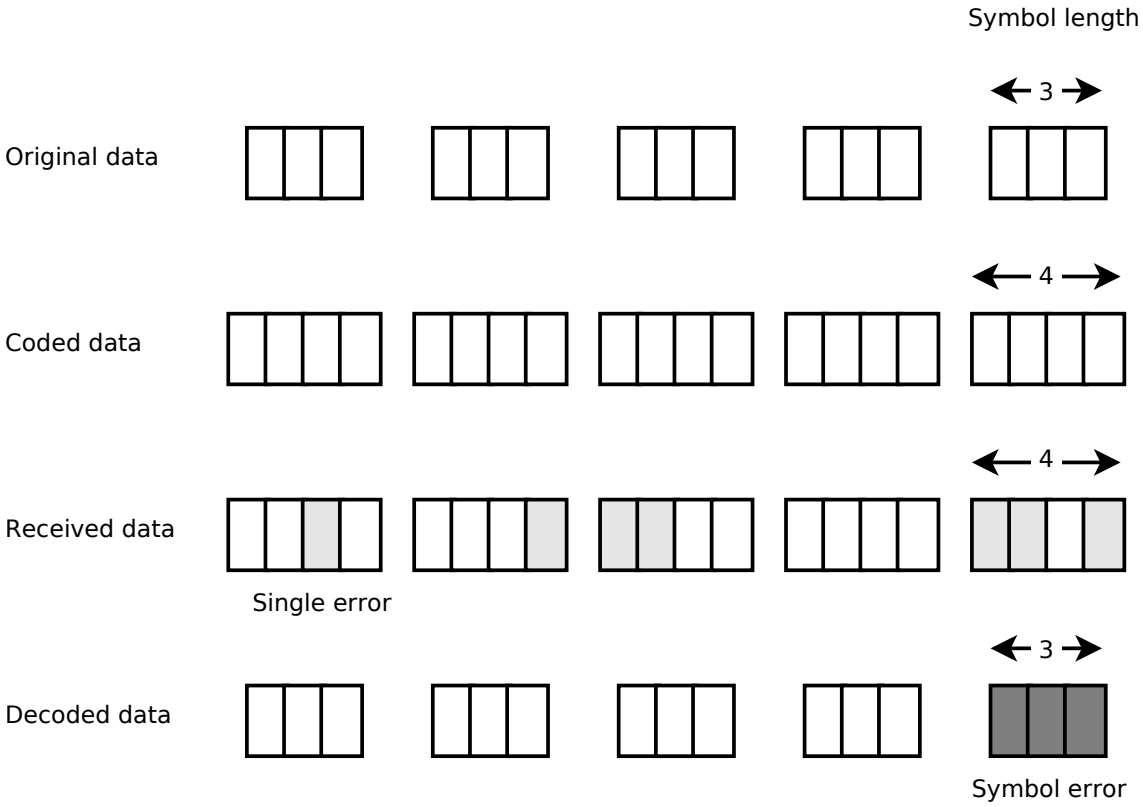

Figure 2.5: Forward Error Correction.

codes with coding rates between 1/2 and 7/8 may then be applied for additional protection, at the cost of throughput capacity.

The applied FEC is designed to provide Quasi-Error Free (QEF) operation, meaning less than one uncorrected error per hour for a 5 Mbps transport stream. However, as the FEC is an in-band coding scheme, it cannot guard against signal blockage or long-term interference when the Signal-to-Noise Ratio (SNR) exceedes the decoding threshold. At medium SNR, DVB's FEC may still provide error detection for the packets by reporting a failure in the packet header. As the SNR decreases, the likelihood of the bit errors effecting important framing packets increases, rendering the detection of individual packets impossible.

This results in arbitrary gaps in the stream that cannot be reliably detected using in-band error protection. The amount of packet loss may be deduced for constant bitrate transmissions if the carrier provides capacity details, but increasing interference will eventually cause the signal carrier to be lost, resulting in full channel outage.

2.2 EXISTING RETRANSMISSION SYSTEMS

Any major error exceeding the error correction capacity of FEC causes information to be irrevocably lost and requires retransmission of the lost data. This presents a problem for DVB networks, as the protocol stack is primarily designed for unidirectional transmissions.

The basis of packet retransmission over bidirectional communication channels is Automatic Repeat reQuest (ARQ)[8], whereby the sender transmits a data sequence, waits for the receiver to transmit acknowledgement (ACK) or negative acknowledgement (NAK) of reception and retransmits any lost data. Except for Stop-and-wait ARQ, which transmits and receives single frames synchronously, the major types of ARQ all rely on numbered frames.

Errors are detected at the receiver by comparing a frame's sequence number to an expected sequence number, either the previous frame's sequence number incremented by one or the size of the frame. ACKs and NAKs containing the next expected sequence number, either one scheduled to be transmitted or one that was erroneously received, are then used to inform the sender of any retransmission requests. Unlike FEC, which solely detects errors in the packets it protects, sequence numbers allows arbitrary gaps of packet loss to be detected, as the cyclically monotonic sequence numbers will define a range between received packets.

The basic DVB stack does not include explicit sequence numbers in any of its fields. A 4-bit continuity counter exists in the transport stream header that provides duplication and loss protection for a 16-packet cycle. For a standard 5 Mbps transport stream with a video substream of approximately 3 Mbps, or 16 000 188-byte packets per second, this still equates to a wrap-around every millisecond. With round-trip times (RTT) between 5-500 ms on bidirectional channels, the counter wraps around an indeterminate number of times during signalling, making it impossible to uniquely address a packet. The effects of a sequence number wrap-around is illustrated in Figure 2.6.

The short period of the counter both precludes the use of them for requesting retransmission of a packet and for reliably detecting gaps in the stream. Any repair system operation on DVB therefore needs to implement both a reliable, cross-peer

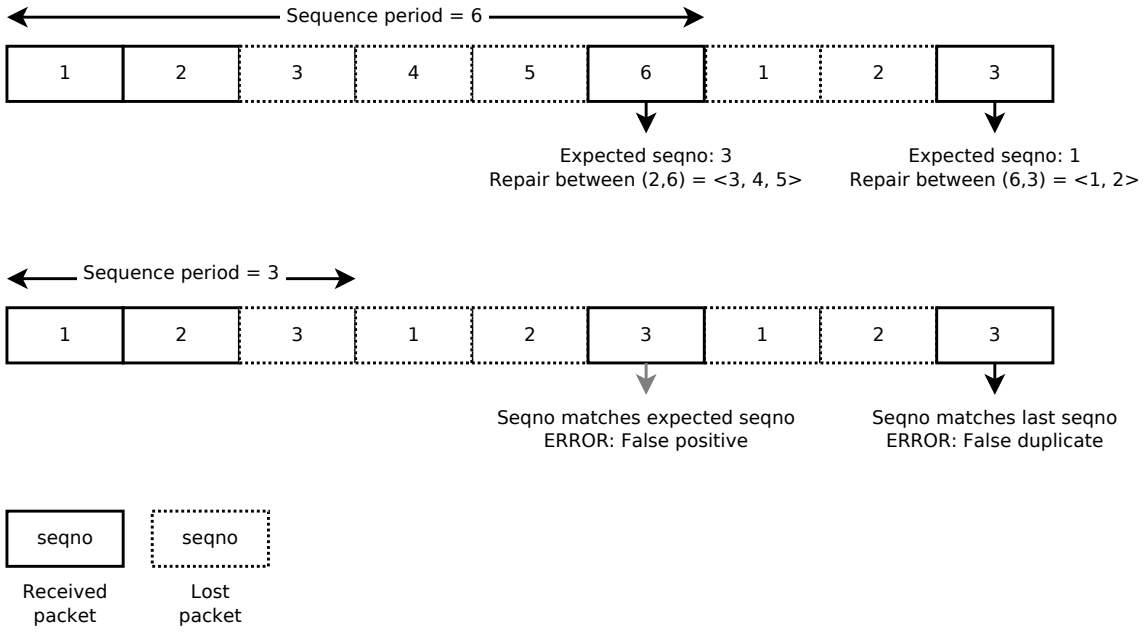

Figure 2.6: Effects of sequence number wrap-around.

addressing scheme for the packets and a way of detecting packet loss.

2.2.1 DVB-IPTV

The DVB standard for IP networks[9] describes an optional retransmission system for broadcasts that have been retransmitted over IP networks. Content Service Providers (CSP) may receive transport streams from a unidirectional source and encapsulate the full stream or a remuxed copy in the Real-time Tranport Protocol (RTP) for transport over IP networks.

RTP is a standardized packet format for transmitting video and audio over the User Datagram Protocol (UDP). An RTP session consists of a stream of RTP datagrams and a separate stream of RTP Control Protocol (RTCP) datagrams, used for providing monitoring and Quality of Service (QoS) for the RTP session.

RTP encapsulation adds a 12-byte header, with possible extension, on top of the UDP and IP headers, to the packets. The payload may consist of an arbitrary, integer number of 188-byte transport stream packets, within UDP and network size limits, as shown in Figure 2.7.

The supplied sequence number is used for reordering of packets and detection of packet loss and duplicates. A timestamp is provided for jitter control at the receiver.

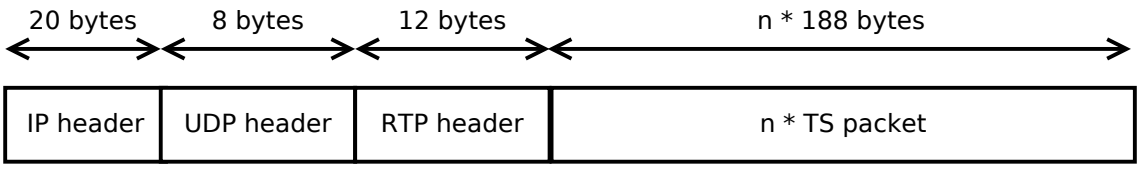

Figure 2.7: RTP encapsulation of TS packets.

It need not be synchronized with the payload time stamps. While the UDP header contains a checksum field for detection corruption, its use is optional.

The RTCP stream may be used to send Compound Sender Reports (CSR) and Compound Receiver Reports (CRR). CSR inform receivers of transmission statistics and CRR inform the sender of reception statistics.

Retransmission (RET) servers may be supplied by the CSP for retransmission of lost packets. A receiver detecting non-consecutive sequence numbers due to packet loss may request retransmission of packets by sending an RTCP feedback (FB) message to the RET server. The requested packets are then retransmitted over a dedicated RTP session.

A RET server that is following a multicast RTP transmission and detects upstream packet loss may prevent excessive FB signaling by sending an RTCP feedforward (FF) message to receivers. The lost packets can then be retransmitted to the receivers using a multicast RTP session.

Encapsulating the original stream in RTP provides the system with performant ARQ capabilities, but also restricts its usage to the range of the relayed stream. Moreover, it requires the full stream to be transmitted over the bidirectional network, which places unnecessary strain on backbone links when the non-IP DVB network already provides a signal.

2.2.2 RELOAD

The 2011 IEEE paper “Using P2P networks to repair packet losses in Digital Video Broadcasting systems”[10] proposes a repair service for digital multimedia broadcast systems based on timeouts. It requires no modifications to the broadcasted content and provides large-scale operation using a peer-to-peer network for servicing repairs.

The “RELOAD” system detects signal loss with a timeout firing after no packets have been received within a certain timeframe, and intermittent packet loss by

inspecting the transport stream's PCR. As the DVB standard defines a maximum interval time of 100 ms between PCR injections, packet loss can be determined when consecutive PCRs deviate from this. A token passing technique is used to distribute chunks of the DVB-T stream evenly over the peers and repair packets can then be requested with the help of a Distributed Hash Table (DHT).

The paper's authors achieved very good results from their experimental measurements. A peer took at most 600 microseconds to join the repair network and 550 microseconds to lookup a repair packet, connect to a peer and receive the packet. While these results are likely to increase by several factors in a large-scale network, they suggest excellent performance for the system.

RELOAD effectively utilizes the content's time stamps as sequence numbers and can perform ARQ for 100 ms chunks of the stream. The system does not provide error detection for individual packet within the chunk, but the error correcting and detecting capabilities of FEC should provide sufficient error control in most cases. The RELOAD system is illustrated in Figure 2.8.

The RELOAD system exhibits most of the traits that we desire in a peer-to-peer repair system, but is restricted to relatively large chunks of data due to the utilization of PCR time stamps. We also discovered a major disadvantage to this in large-scale multi-transmitter networks during our testing and assessment, which would limit the efficiency of the system to a single DVB transmitter's range.

The DVB standard suggests that the PCR time stamps be generated from the multiplexer's internal clock and our testing found that these clocks may have large offsets in comparison to each other. The RELOAD system's logic that depends on a shared time base can therefore only function within the range of one transmitter.

Note however that this is specific to the implementation of the DVB network. Later testing showed a shared time base between multiple transmitters. This suggests that the network has been modified to take input from a source that centrally multiplexes the content. We discuss this further in the feasibility testing and assessment sections.

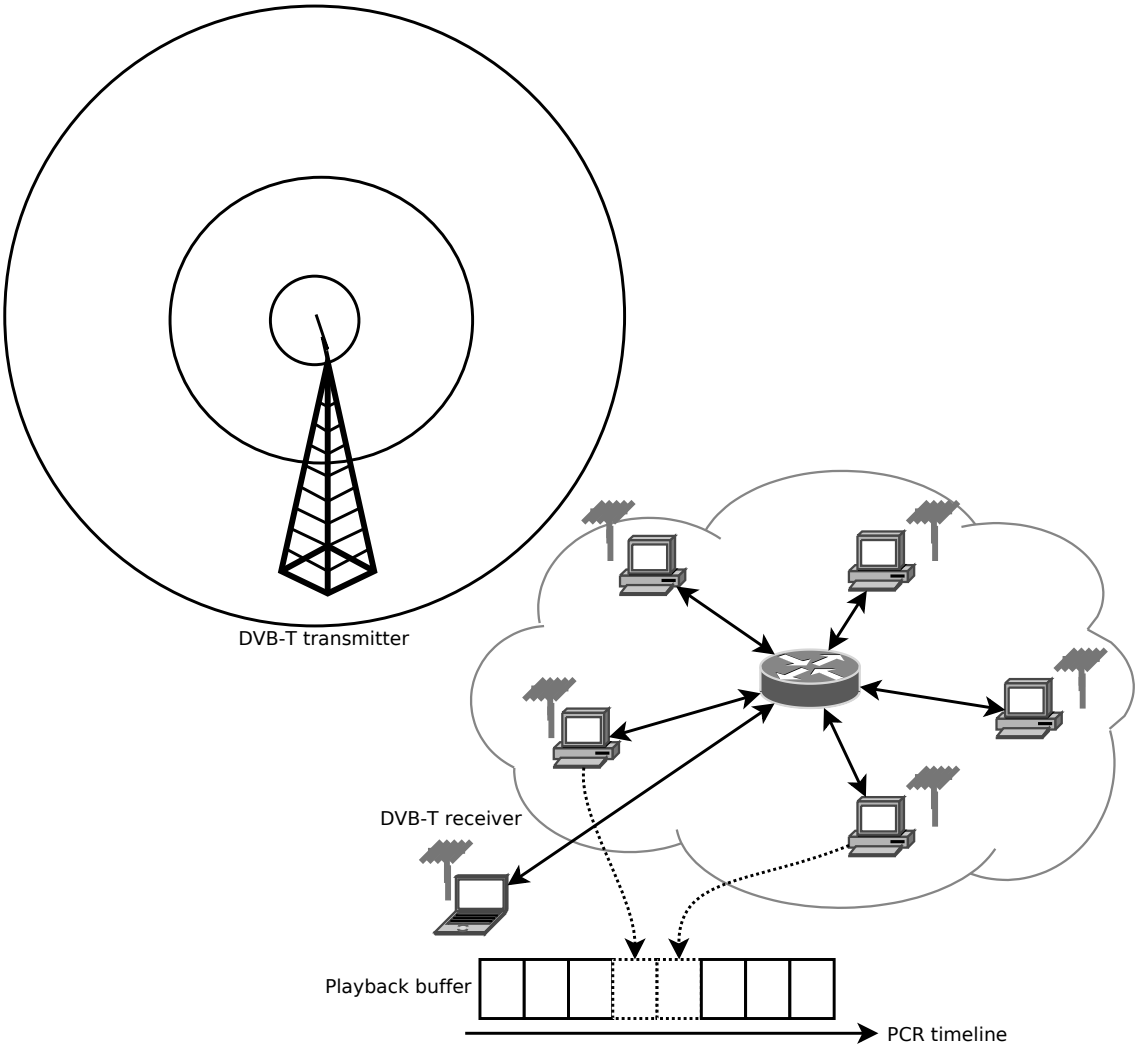

Figure 2.8: Lost chunks of a DVB transmission may be repaired using the RELOAD P2P network.

3. PROPOSED SYSTEM

The system that we propose is based on peer-to-peer synchronization of the stream payload instead of any form of inserted or synthetic sequence numbers. The methods should allow for byte-perfect error detection with probabilistic error correction that increases with the number of peers.

The error correction is only probabilistic because of the system's peer-to-peer nature. Each peer may receive its data over an unreliable link and the system can only detect differences between the peers' data. If all peers suffer from the exact same errors, there is no way of determining what the missing data is. The system could attain perfect error correction if any peer has a perfect link to the source data, but as the system does not provide any authentication of peers, we must rely on majority voting to prevent poisoning of the data by a malicious peer.

The system takes advantage of probability theory's rule of multiplication for independent events:

$$\Pr\left(\bigcap_{i=1}^{n} A_i\right) = \prod_{i=1}^{n} \Pr(A_i), \tag{3.1}$$

which states that the probability of all events, errors in our case, is the product of the events' probabilities. As the probability of an error event is typically a lot less than one, the probability approaches zero as the number of independent events increases.

We first present the theory and basic techniques behind the system, after which we present the results of feasibility testing. Finally, we describe the design of a full solution.

3.1 THEORY

Without a method of reliably addressing distinct pieces of the DVB stream, repair of any error is difficult. A receiver can detect the errors through coding and timeouts,

but only determine the errors' locations in the receiver's own buffer.

As the data does not contain adequate sequence numbers, which would provide synchronization, this leaves us with the distributed systems problem of achieving distributed shared state. Even when the content is transmitted at a constant bitrate to a all receivers, with the corresponding element arriving almost exactly at the same time to each receiver's buffer, relaying information regarding that element is effected by transmission lag. As shown in Figure 3.1, any lag longer than the time between two DVB packets will cause a retransmission request of the last received packet to retransmit the wrong packet.

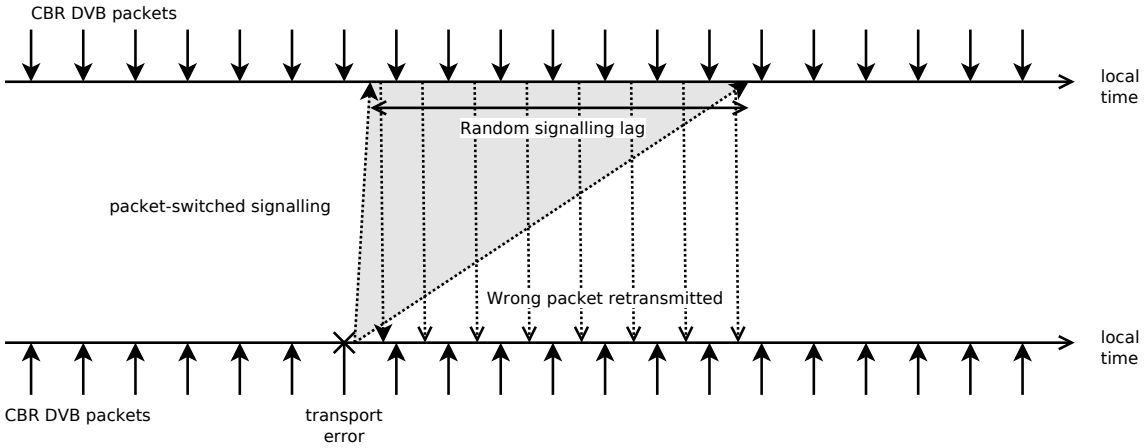

Figure 3.1: Retransmission of the last packet over a link with lag. The repair request can fall anywhere inside the grey area due to the nondeterministic transmission lag.

As the lag is effectively nondeterministic in a packet-routed network, we cannot reliably determine the offset caused by the lag using any kind of signalling. Attempting to signal the arrival of a DVB packet in a stream with a constant bit-rate of 5 Mbps using a signalling channel with an average RTT of 50 ms may cause an erroneous offset of around 100 000 packets.

3.1.1 DATA DIFFERENCING

If the signal carries enough information to identify a packet, we can attempt to determine an offset between multiple receivers' streams by comparing other packets to that information. This may require the traversal of a large buffer of packets, but with modern low-latency memory, the time taken should be trivial compared to the network latency. The synchronization of a single pair of packets would then allow

us to address arbitrary packets in both streams using relative offsets.

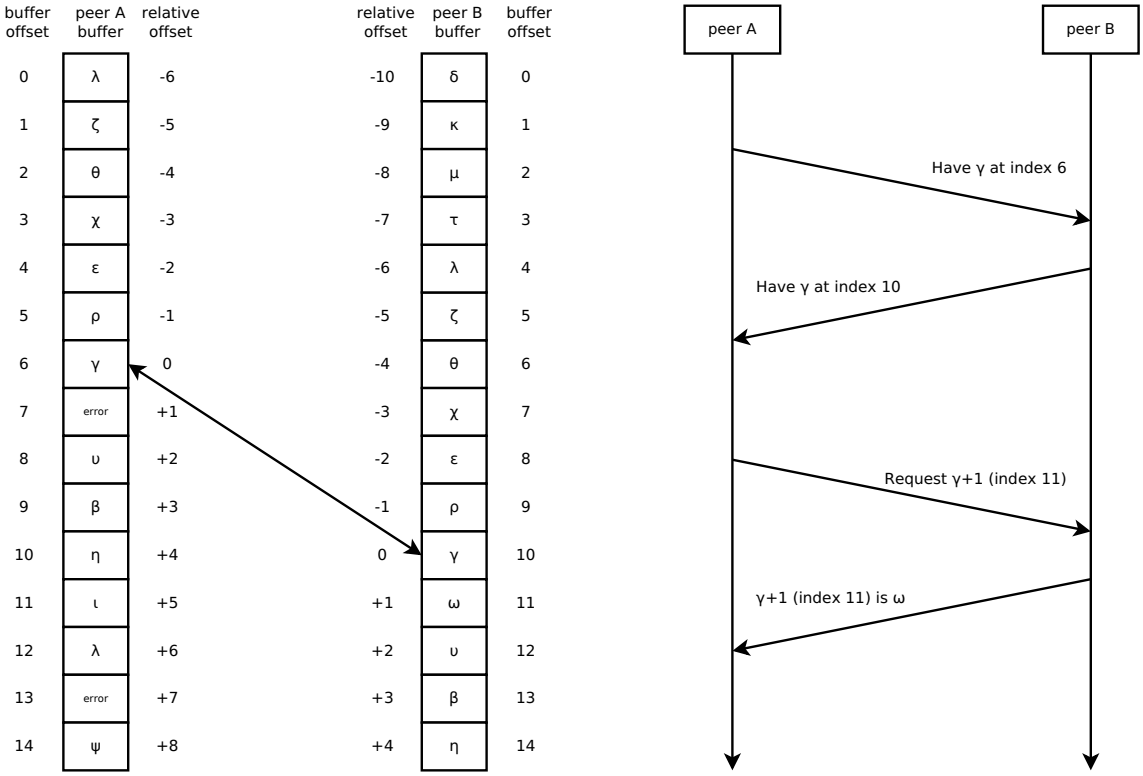

Figure 3.2: A single synchronization point allows relative addressing of any point in an unlimited sequence.

Figure 3.2 shows a simple method of synchronizing on payloads. It assumes that each element is unique, which will quickly fail if e.g. peer A attempts to repair the error after the second λ by synchronizing on the λ.

The likelihood of each element being unique in an effectively infinite stream of data approaches zero. In the DVB domain, we can assume that most peers will operate on a very similar subsequence of the stream due to the programming being prescheduled and transmitted at approximately the same time. A full block of content is unlikely to be repeated within that subsequence, but single access units and even short sequences of audiovisual data may be repeated. Any synchronization algorithm operating on this data must therefore be resistant to repeated elements.

Assuming the content is mostly unique, with low periodicity, the task of detecting an offset becomes a string matching problem. By treating the packetized content as an alphabet, we can attempt to find the offset, or shift, of one receiver's content string in another receiver's content string. This would allow receiver's to synchronize

their buffers to each other and allow retransmission using relative addressing. Under perfect conditions, synchronization could be done as a simple substring match. The transmissions' payloads may be shifted or different lengths, but a match could be found by direct comparison.

String matching is an old problem, with efficient algorithms dating back to the beginning of computer science research. The best known algorithm is likely the Boyer-Moore string search algorithm, which was developed in 1977.

As the transmission may contain errors, we cannot simply attempt a string match of the full length, but must select substrings that are error-free and common between the peers. This problem is known as the longest common substring problem in computer science. The algorithms used to solve the problem generally rely on suffix trees, a tree containing all the different suffixes of the strings.

This is efficient for cases where we want a full substring match, but fails when we instead want the best possible match for mismatching strings. The content that this system will operate on will likely have significant numbers of errors. This will result in strings that are for the most part similar, but have randomly occurring mismatching elements. The strings may contain only short uninterrupted substrings, but have many matching substrings in succession.

The problem then becomes one of finding the Longest Common Subsequence (LCS)[12]. Unlike a substring, a subsequence is a sequence of elements where each element of the subsequence is present in the same order as in the parent sequence, but with gaps allowed. The LCS of two strings is thereby all elements that appear in both strings in the same order. When both sequences have random errors, the longest common subsequence offers significantly better matches than the longest common substring, as illustrated in Figure 3.3.

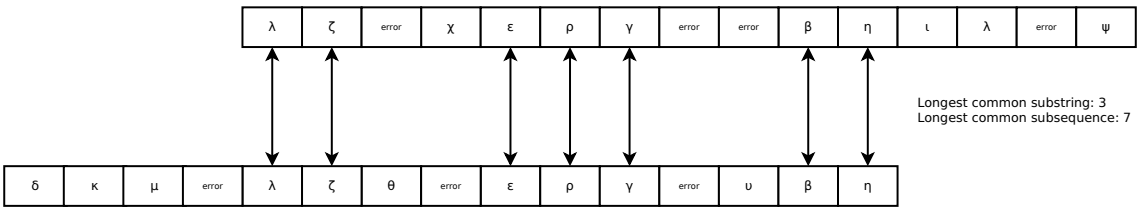

Figure 3.3: Longest common substring and longest common subsequence.

From an LCS, we can also determine all elements of the original string that do not

appear in the LCS and so obtain the differences between the strings. The differences can be used to generate a Shortest Edit Script (SES), which defines how one can convert one of the strings to another using element insertions and deletions.

Like the problem of string matching, determining the LCS and SES is an old computer science problem with mature algorithms. The algorithms are extensively used for finding similarities between Deoxyribonucleic acid (DNA) sequences in molecular biology, versioning of files in source control systems and deduplicating data in file systems.

The time complexity of a generic LCS approach with N input sequences is quite high at

$$O\left(N\prod_{i=1}^{N}n_i\right). \tag{3.2}$$

The length n_i of the sequences is likely to be thousands of elements, but we can assume a single-digit N, as the probability of even a few independent error events occuring simultaneously is low.

Lower time complexities can be achieved when taking into account further limits on the system. The resulting techniques are generally classified as data differencing or delta encoding techniques. The most popular implementation is likely GNU diff, which compares texts on the line-level and outputs all differing lines.

The paper "Improved string reconstruction over insertion-deletion channels"[13] proposes an $O(mn(logn)^2)$ solution for m variations of the same n-length string that closely matches our problem. Modeling of the domain and its interaction with the existing error control systems may further restrict the complexity of the problem.

Along with providing relative addressing through offsets from known synchronization points, these techniques provide error detection through data differencing that can potentially be more effective than current error detection methods. The actual payload is being checked at a binary level unlike sequence number- and checksum-based systems that solely operate on metadata.

3.1.2 DISTRIBUTION

Any type of content matching assumes the availability of each receiver's content at a shared location. In a peer-to-peer network, this still leaves us with the problem of

distributing all content to each peer. We only require a subset of the content at a time, but that subset must constantly be updated as new errors cause synchronization to be lost between peers. Restreaming the full content to each peer would be prohibitively expensive for any fixed-capacity links and be highly inefficient when only a few errors occur in each trace.

We therefore require a way of transferring the pertinent information as compactly as possible. There are a multitude of ways to compress audiovisual content and the DVB data is already highly compressed using Moving Picture Experts Group's (MPEG) MPEG-2[7] or MPEG-4[11] compression encodings, but these methods attempt to encode significantly more information than we need. The algorithms used to determine LCS and SES only require the ability to perform equal/unequal tests on the elements in the alphabet and that the sequences' orders are preserved.

Hash functions[14] attempt to preserve data equivalence by applying a deterministic algorithm to the data and returning a hash sum. The reductive properties of a hash function does mean that it maps a large data set into a smaller data set and may therefore produce hashes matching multiple, inequivalent packets. This is referred to as a hash collision and is shown in Figure 3.4 as multiple mappings ending at the same element. This necessitates that we do not synchronize on single elements, as this may give false positives, but on longer sequences.

Each hash sum can substitute an arbitrary amount of data, allowing a high bandwidth stream to be reduced into a representation that can easily be transferred over low bandwidth networks. Each hash sum will be positioned in the same order as the original data that it represents and provide adequate uniqueness for the data difference algorithms that operate on it. The data differencing algorithms that operate on the resulting stream already assume an unknown shift in the location of the elements due to the original DVB networks' transmission lag, so the stream of hash sums do not require low-latency distribution to other peers.

This makes the system robust for both high-latency and large-scale networks. Each stage may add some additional latency, but the receiver may buffer the content until it has been fully repaired and only display the result when ready. While this is suboptimal for the real-time transmission that DVB was designed for, we expect

that consumers forgive the added display delay as they have become accustomed to it through the use of buffering Internet video.

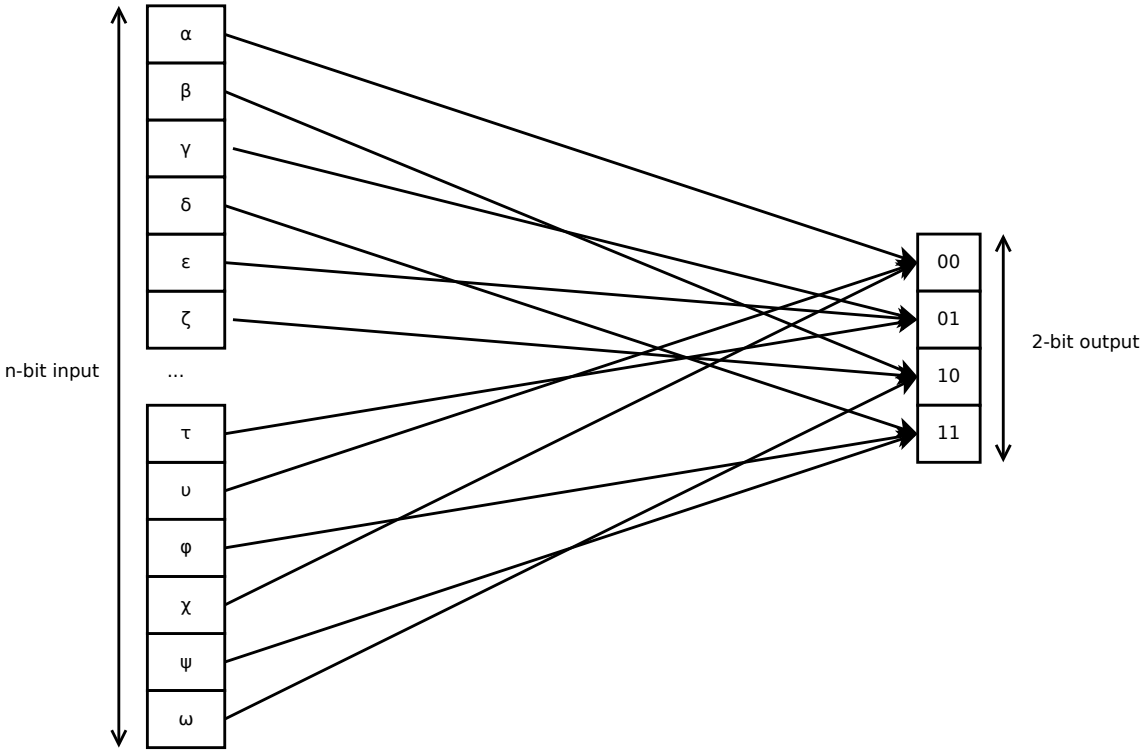

Figure 3.4: A hash function maps a large data set to a smaller data set.

3.2 METHODOLOGY

Hash functions and sub-sequence matching are both mature techniques that have countless implementations in real-world usage, but building a fully-functioning repair system with them may prove highly inefficient or even impossible. To examine if the proposed solution had any possibility of function on real-world data, we attempted to implement a simple proof of concept.

We conducted feasibility testing in the Finnish terrestrial and cable DVB networks in an attempt to prove the feasibility of the idea. Initial testing in a single region using dvbsnoop, a DVB analyzer, to record DVB transmissions from two different antennas and comparing them using GNU diff, an SES/LCS solver, was positive. We modified a copy of dvbsnoop to output Pearson's hashes for TS payloads and 32-bit Cyclic Redundancy Check (CRC32) hashes for PES payloads, and were able to synchronize the traces effectively without false positives due to hash collisions. Any

modifications to the payloads resulted in large changes in the hash values, promising good error detection for the technique.

For multi-region testing we set up four DVB signal sources:

- A Linux laptop in Hindersby using a Universal Serial Bus (USB) tuner in Digita's terrestrial network, Pernaja transmitter.
- A Linux desktop in Espoo using a Peripheral Component Interconnect (PCI) tuner in Digita's terrestrial network, Espoo transmitter.
- A Windows 7 desktop in Tampere using a USB tuner in Tampereen Tietoverkko's cable network.
- A Windows 7 desktop in Turku using a USB tuner in Digita's terrestrial network, Turku transmitter.

Analysis conducted on traces from multiple regions showed positive matching between the payloads' hashes, but with a higher error rate than indicated by the packets' TEI flags. This suggests that not only the packet headers, but the actual payloads have differences between regions. We therefore require a method of detecting and removing these differences for the sub-sequence matching to work, effectively providing canonicalization of the data.

The combination of dvbsnoop and diff proved unwieldy for automated testing, so we initiated the developement of a custom proof of concept of the system. A proof of concept was developed between Fall 2011 and Spring 2012 as a set of C programs that accepted TS input and fed their output to the following stage's program. This separation, based on UNIX philosophies, allows for easy insertion of filters between components. This allowed the system to accept TS input from arbitrary programs and distribution of the stages over a network using the netcat program.

Our development and tests were cut short due to a modification in the Finnish DVB network in Spring 2012. The transmitters that earlier had transmitted independently multiplexed content, with unique time bases, started outputting synchronized content with a common time base.

We were unable to obtain empirical results of the system before the change, but the early development did yield some insight into the challenges that the system

would face. The design and assessment therefore still assumes that the characteristics of the original network are present.

3.3 DESIGN

The design for the proof of concept consists of three main components: a hasher, a comparator and a reconstructor. The hasher's functionality can further be split into three stages: chunking, canonicalization (c14n) and hashing. The process is illustrated in Figure 3.5.

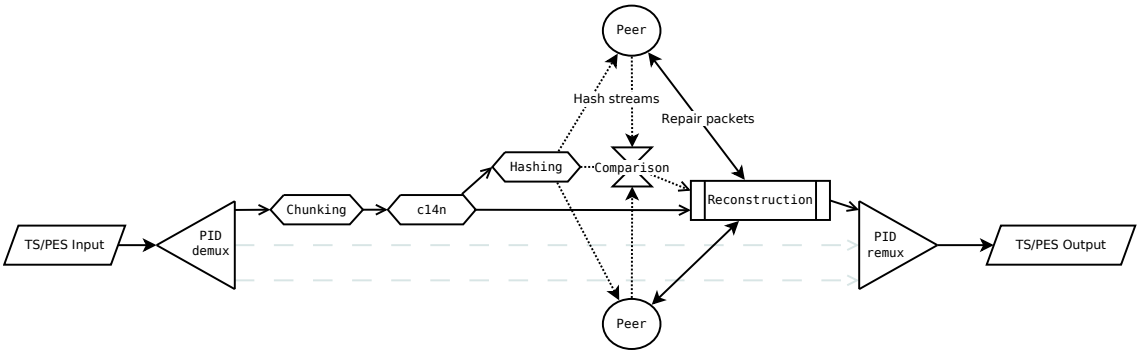

Figure 3.5: Flow graph of the repair system.

The DVB input stream is fed into a demultiplexer that separates the stream into its individual sub-streams. Each sub-stream is deterministically chunked, has any extraneous padding or stuffing bytes removed and is hashed.

The resulting stream of hashed chunks can then be transmitted to any peers participating in the reconstruction. The peers transmit their own hash streams back to the comparator component, where the streams are synchronized, compared and a majority vote decides which hash is accepted. This majority vote allows conflicts to be resolved to the likeliest correct solution.

Local hash elements that do not match the voted solution are discarded and a repair request is sent for the replacing chunk. Finally, the reconstructor uses the output of the comparison to reconstruct a repaired data stream using the original payload data and any repair payloads returned by the repair requests.

3.3.1 PREPROCESSING

As specified in Section 3.1.2, the primary reason for hashing the DVB stream is to reduce the required bandwidth to transmit it to other peers. National DVB networks cover thousands of miles and bidirectional communication between peers may have a RTT averaging 50 ms, with some connections averaging in the high hundreds of milliseconds. Consumer connections are generally asymmetric, with uplink bandwidth between a few hundred kbps to a few Mbps and downlink capacities between one and a dozen Mbps.

We chose a target link of 2 Mbps down and 1 Mbps up, the capacity of a mid-to-low-end consumer link. We therefore require a minimum reduction factor of at least 5:1 to retransmit a hashed representation of a single 5 Mbps DVB stream. Consumer links rarely provide dedicated capacity, so an actual minimum is likely to be at least 8:1. The link will be further stressed by the full-size repair packets required to correct errors. Repairs would require a 5 Mbps link when full signal loss occurs, but we can allow for buffered reconstruction in such a case. A viewing delay is preferable to full outage and we expect consumers have become accustomed to delayed viewing through the proliferation of time-shifting DVRs and Internet media.

Hash functions come in many forms and are still subject to ongoing research. Cryptographic hash functions try to make it difficult to generate a message with a known hash sum, a rolling hash allows for efficient rehashing of a substring window that moves through a longer string and various special-purpose hash functions function efficient in single domains by taking advantage of some characteristic of the data.

An example of this is perceptual hashing[15]. Perceptual hash functions extract information from audio and video data that is relevant to the human perceptual model and generate a hash sum from it. The information can, for instance, be beats extracted from a song, motion vectors in a video or simply a downscaled and normalized representation of an image. The primary purpose of the information extraction is to make the hash function robust by removing any data that is irrelevant, like minor errors and transformations, while keeping the hash function's uniqueness, so that a perceptually different input results in a different output.

Perceptual hashing fits our domain, but the algorithms primarily focus on robustness, as they are mostly used in detecting copyright violations, where a malicious human may attempt to break the hash function. This may result in matches where the repair packet is from the same content, but a degraded source, and would produce an unwanted quality discontinuity. With adequate smoothing, such a system could provide hierarchical error correction, but for the purposes of this thesis, we shall focus on a simpler design.

A rolling hash is the preferred method of data deduplication systems, as the data does not have to be in discrete blocks before hashing. Instead, the algorithm traverses through the data one byte at a time, while constantly recomputing and comparing the hash sum of a fixed-size window of data. Data may be injected or deleted at arbitrary locations without the comparison failing. This closely matches the error model of a switched network where errors may occur as data corruption, loss and duplication.

We chose to simply support any general-purpose hash function. The content we operate on will be transmitted as FEC-protected fixed-size TS packets, with higher layer protocols available. This gives the data an inherent structure that we can use to deterministically chunk the data into suitable blocks for any hash function. As the chunking and hashing logic is fully deterministic, each peer can independently perform the operations and receive the same answer for the same data.

The data may come from multiple sources and, as such, may have minor differences in the structure and associated metadata that have no effect on the content. General hash functions produce an aggregated representation of each byte of the data, so we must explicitly remove any data that is irrelevant. We accomplish this canonicalization by ignoring any insignificant protocol headers and removing any unwanted padding of the packet payloads.

As we need information of the structure of the data, both sanitizing and chunking the data requires implementing a decoder of the DVB stack. As every layer increases the complexity of the implementation, it would be preferred that both functions could be performed at the lowest possible layer.

From Section 2.1, we know that the first layer is comprised of TS packets. TS

packets have an average payload size just under 184 bytes. With a one-byte Pearson's hash, this would produce an approximately 180:1 hash ratio. Initial tests at hashing at this layer produced good results, but failed whenever a time stamp was defined in the header. The time stamps were injected at different offsets in different regions. This caused the payload to shift, producing differing hashes even when the content of the stream was the same. Figure 3.6 (a) illustrates how the payload was shifted when an adaptation field containing a time stamp was present. A full listing of the diff can be seen in Appendix A.

By going a layer higher, to the PES level, we got access to the sequential payloads of multiple TS packets. The size of PES packets is variable, but mostly between 1024 and 8192 bytes. Transitioning to a 4-byte CRC32 hash to be safe, we get an average 4096:1 hash ratio.

Initial tests with chunks hashed at the PES level produced near-perfect results between two DVB-T sources, but failed between DVB-T and DVB-C sources. While most of the data was identical, One source appended padding zero-bytes to the video slices on word boundaries. Audio frames had similar padding added, but also showed a substantial shift in the content without an obvious underlying cause. Listings of the video and audio diffs can be found in Appendices B and C.

We did not attempt to implement full parsing of the elementary stream layer, as that is outside the scope of this thesis, but we did identify the audio payload as fixed-size 670-byte MPEG Layer 2/3 audio frames. The audio frames were shifted by a constant amount, but were otherwise in the same order. Figure 3.6 (b) illlustrates the shift in audio frames. The fixed-size frames allowed for easy chunking of the audio payload and a decent 670:1 ratio for the hashing.

The padding bytes in the video payload were simply removed by filtering out zero-bytes, after which we hashed the full PES payload. While this may result in false positives in the comparison component, the occurrence of zero-bytes outside of padding is small enough that the results should closely match an implementation that fully parses elementary stream access.

As illustrated in Figure 3.7, the hashing component produces a hashed stream that represents the elementary stream payload, with TS and PES framing removed.

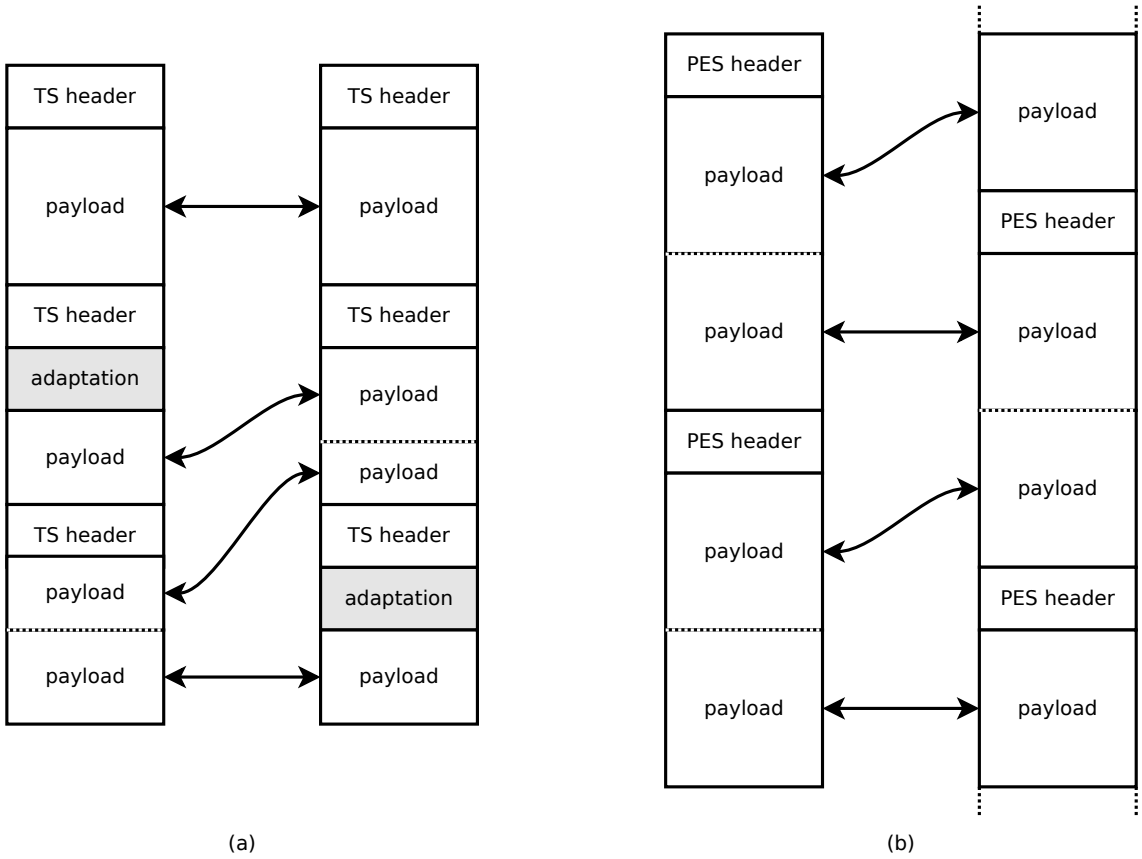

Figure 3.6: Payload shifts caused by inserted headers. (a) Adaptation field insertion in transport streams; (b) PES header insertion between audio frames.

Each hash represents a deterministically decidable chunk of the stream and is listed in the same order as the original content.

Each peer performs the same operations on their input stream, producing a hash stream, and exchanges it with other peers. As this takes place on a bidirectional network, we can rely on existing protocols to guarantee reliable transmission. While these increase jitter to the transmission, the synchronization methods used in the next stage already need to address the inherent jitter of the packet-switched network, so it should not affect the outcome.

Our tests assume perfect autoconfiguration of the exchange, but a real implementation will require each stream to be matched in some way. Each input stream contains multiple substreams, which will be hashed individually. The combined information stored in the transport stream's PSI tables and the Electronic Program Guide (EPG) stored in the Event Information Table (EIT) should however provide sufficient matching data for the streams.

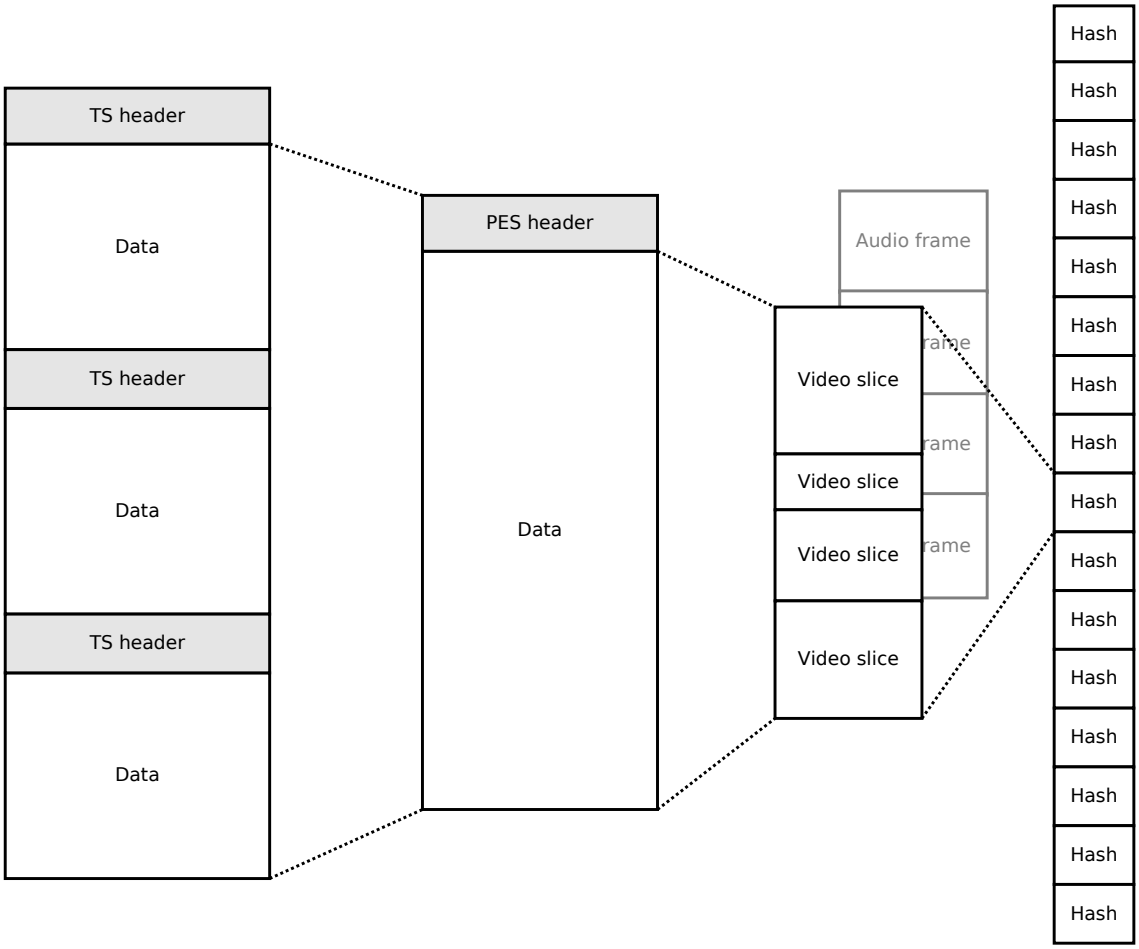

Figure 3.7: An overview of the hashing component.

3.3.2 ERROR DETECTION

The exchange will provide each peer with a set of unsynchronized streams that it feeds into the comparison component, illustrated in Figure 3.8. The propagation delay of the DVB signal, processing and transmission delay of the hash exchange component all add to the offset of the signals, so we must support a reasonably large search space. This will primarily be a computational problem, as the large chunks and small hash elements take little storage space. Our tests used a 4 MB buffer, which is sufficient for a million 32-bit elements, far more than necessary.

The limited variations of a 32-bit hash may cause false positives due to hash collisions. As we want to retain a high hash ratio and high granularity for better error bounding, using a larger hash would not be beneficial. We instead require a match of at least 10 sequential elements. As the algorithm searches for the match one

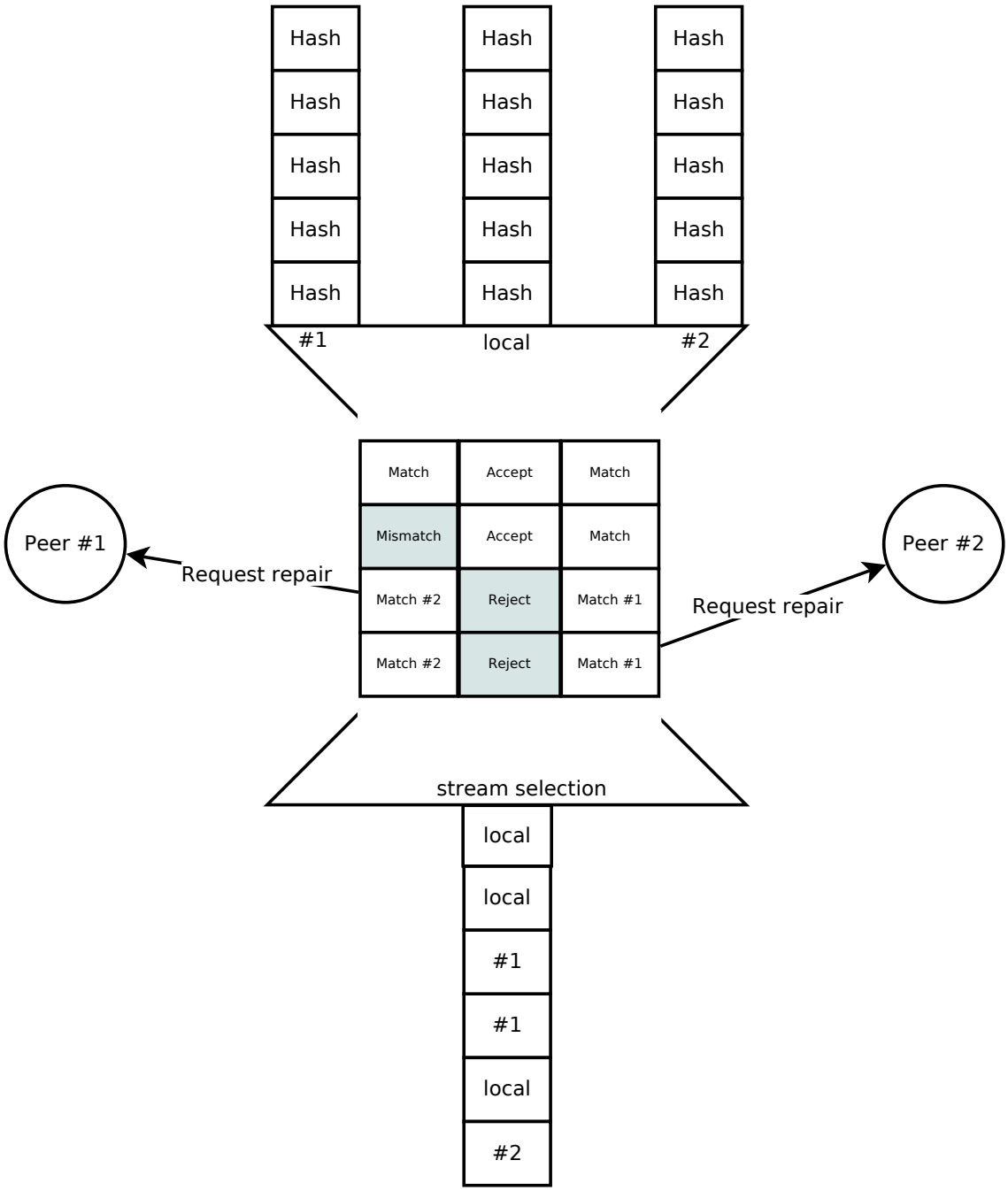

Figure 3.8: An overview of the comparison component.

element at a time, this retains a granularity of 32-bits, but prevents false matches by requiring a 320-bit sequence of bits to match. The downside of this method is that all streams must have periods of high SNR for synchronization to occur.

Another method would allow synchronization as long as n elements in a sequence are matched, regardless of potential mismatches in the sequence. This would better match signals with low, but constant, SNR. The mismatching gaps should however

be limited to reasonable lengths to prevent false matches to occur, as the unbounded stream may contain a near-infinite variation of sequences.

Once a synchronization point has been found, the error detection component can use a simpler algorithm to step forward each stream in unison until a mismatch is detected. Packet loss will require resynchronization, but is likely to be within a small window. We will however assume a LCS algorithm is used and that any differing elements are returned as a SES of inserts, deletions and substitutions.

The system will need to detect three error conditions: packet loss, packet corruption and packet duplication. Support for DVB streams sourced from packet-switched networks may further require support for packet reordering.

While substitutions can be verified as corruption errors by checking the TEI flag of the corresponding TS packet, the correct case for the ambiguous dropped and inserted packets cannot be easily determined if continuity counters do not detect discontinuity. Comparing only two streams, an insert in one stream is equivalent to a delete in the other stream. The likelihood of an ES level chunk being inserted is low, but it may however occur. For this reason, we will require at least three different streams for these cases, with conflicts solved using a majority vote.

Out of order packets can be identified as a special case of an insert shortly followed by a delete or vice versa. Although this case is possible to optimize by storing the inserted packet, these cases can be safely treated as a missing packet by ignoring the insert and requesting a repair packet for the missing element. The majority vote will resolve the case to satisfaction.

Repair requests should be sent as soon as possible to minimize the repair delay. The simplest solution is to request repairs from the peers participating in the peer exchange, as the streams are now synchronized and any element can be addressed using offsets relative to the start of the exchange session or any other element uniquely identifiable between the peers.

Future implementations may find the use of some kind of distributed index beneficial. The majority vote should have resulted in a probabilistically canonical stream of hashes. It should therefore be possible to index the payload using a substring of the canonical hash stream, as long as the substring is long enough to be unique.

This method would allow more peers to participate in the repair process without having to exchange streams for the purpose of detecting errors.

If the received repair packets are queued in the same order as they were requested, we now have a set of buffered streams that contain all data required to reconstruct an error corrected transmission. The comparator component only needs to output a source selection stream to the reconstructor component so that it can reinterlace the payloads.

3.3.3 ERROR CORRECTION

The reconstructor component, illustrated in Figure 3.9 will operate on the output of the comparator, the original data stream and the payloads of the repair packets. Its main function is to simply reinterlace the chunks. However, as the system only compares payloads and disregards the headers that may significantly differ between regions, reconstruction of the repaired content may require extensive rewriting of these headers.

Time stamps need to be shifted to a common timeline and may need to be interpolated and inserted in cases where the maximum gap has exceeded 100 ms due to a repair from another source. The time stamp reconstruction will need to be deterministic, as it is used for intrastream synchronization of the substreams. The easiest solution should be to simply use the local source's timeline, as it is shared between substreams.

This requires rewriting at both the TS and PES level, as they both carry time stamps in their headers. When remuxing TS, a repair may also modify the interleaving of substream packets and negatively effect performance of decoders unless intelligently reinterleaved. It may therefore be beneficial to remux into a PES-level container like Program Streams (PS) unless TS output is required.

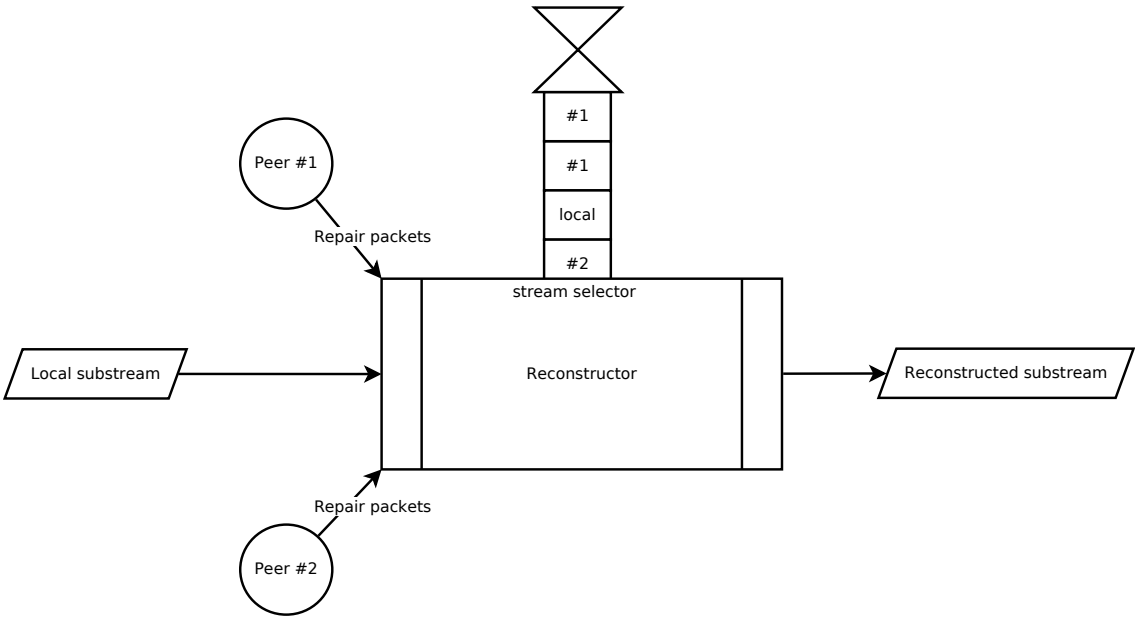

Figure 3.9: An overview of the reconstruction component.

4. ASSESSMENT

The proposed system uses techniques that are new to the domain of retransmission services. It may be interesting simply as an area of research, but for it to provide real practical value, it must provide greatly increased efficiency in solving a problem or novel use cases.

In the following sections, we will attempt to assess the system in relation to the two existing systems presented in Section 2.2. We will refer to the optional retransmission system in the DVB-IPTV standard as "DVB-IPTV" and to the peer-to-peer system defined in the 2011 IEEE paper as "RELOAD". While we will try to assess the systems objectively, this thesis primarily focuses on consumer-run peer-to-peer solutions, so we will prefer systems that can operate without expensive infrastructure or provider assistance.

4.1 COMPLEXITY AND IMPLEMENTATION DIFFICULTY

The complexity of the proposed system is significantly greater than that of the other two systems when taking into account worst case scenarios. Average and best case scenarios may however be reduced to similar logic: "request a retransmission if the received value does not match the expected value".

The systems using cyclically monotonically increasing sequence numbers handle many scenarios better as they can deduce the missing values using extrapolation. The error detection can therefore be seen as having $O(1)$ complexity. The proposed system based on data differencing must first determine the correct expected value from multiple unreliable sources and can in the best case only be $O(n)$, where n is the number of sources required to detect all errors. The complexity quickly increases to $O(nm)$ as synchronization is lost and the system must search for reference points in m elements of each source's value stream.

Both the canonicalization and reconstruction stages also require extensive knowledge of the protocol stack and domain so as to produce valid output. Canonicalization requires that the input be demuxed, sanitized and chunked deterministically. This can be accomplished by using the existing codebases of media decoders, but may bring along bloat that would make the system inviable for infrastructure use.

Reconstruction of the repaired stream may similarly take advantage of existing media encoders. One should however attempt to use nondestructive remultiplexing logic instead of transcoding logic so that the output can be reused within the system. Existing encoders may not have this constraint.

The hashing and error detection stages do not require domain-specific knowledge and can use generic algorithms. This may however not be optimal as there are only a few protocols lacking sequence numbers and the logic may greatly benefit from domain-specific optimizations.

4.2 OVERHEAD AND NETWORK EFFICIENCY

The overhead of the presented systems cannot be directly compared as they do not assume similar networks. DVB-IPTV effectively has an overhead of over 100% as the signal is encapsulated and relayed. While this is not fair in a network designed to be solely accessed using IP, for a hybrid network it is a great disadvantage.

If one ignores the overhead of relaying, DVB-IPTV has the least overhead as it only needs to retransmit single packets. The system also specifies a multicast retransmission solution that greatly reduces total overhead in some cases. However, these take advantage of the encapsulation, which is only possible by relaying the transmission, so for our purposes we will not ignore the relaying overhead.

The RELOAD system primarily uses a non-IP DVB signal to feed each receiver, so does not incur the relaying overhead. Retransmissions require only a DHT lookup and transmission of a configurable 1 second chunk of data, making it the most network efficient of the three systems.

The proposed system has the overhead of the N hashed streams from each contributing peer and a domain-determined retransmission chunk size. The hashing should reduce each stream to a small fraction of the original stream, so the com-

bined overhead should never exceed the relaying overhead of DVB-IPTV.

4.3 CORRECTNESS

The primary advantage of the proposed solution is its correctness when taking into account an unreliable upstream link. The system assumes that each peer has an unreliable upstream source and takes advantage of the geographical distribution of peers to detect interference.

The handling of transmission errors upstream of the relayer is explicitly left undefined in the DVB-IPTV specification. Service providers may position the relayer at a location with high SNR and accept any remaining errors or obtain a reliable link to the upstream source. This would be equivalent to having a master peer in the proposed system, which would significantly reduce the complexity and similarly provide perfect correctness without the overhead of relaying the transmission.

The RELOAD system does not perform any actual error detection, outside of the demodulator's FEC, instead relying solely on loss detection for triggering retransmissions. With a sufficiently high SNR, the FEC will take care of any transient errors and loss will quickly be repaired by the low-latency retransmission system. However, both the original signal and the retransmitted chunks may have a low SNR and cause decoding errors. This could be mitigated in a revised system by rating peers with a quality metric and using error indicators provided by the FEC or a media decoder to trigger retransmissions.

4.4 SPECIFIC ADVANTAGES

A unique advantage of the proposed system is its ability to perform canonicalization of independently multiplexed transmissions. This allows the system to source from a large network of distributed transmitters, even when these transmitters generate differing metadata.

A DVB-IPTV network at a similar scale would have the relayer as a Single Point of Failure (SPoF) and use expensive backbone bandwidth for the initial relaying. The system's RET servers may be independently distributed throughout the network, so repair latency and total backbone bandwidth would still be acceptable in most

cases.

The service provide may segment the network by providing multiple relayers, each receiving their stream from a non-IP DVB source. This would mitigate the load on the backbone network and contain the failure of a single relayer to its segment. From the point of view of a commercial service provider, this allows for simpler provisioning of resources and better failure management. However, this would decrease the efficiency of the network as servers in one segment could not assist receivers in another segment due to the content and RTP sequence numbers not being synchronized between segments. Solving this problem would essentially require a similar synchronization stage as defined in the proposed system.

The RELOAD system suffers most in a large heterogeneous network as it depends on the PCR time stamps inserted at the TS level for determining packet loss and the constant transmission latency for determining signal loss. The variable latency may be worked around by synchronizing on protocol headers, but an offset in the PCR time bases would require payload-based synchronization as defined in the proposed system. Without a common time base, the peers in a RELOAD system are confined to the range of single transmitter.

4.5 SPECIFIC DISADVANTAGES

The low-layer conditional access system defined in the DVB standard presents a problem to our proposed system, as the encryption scrambles and obfuscates any underlying structure. This makes canonicalization impossible without first decrypting the data. As the receivers do not have the original encryption keys, they cannot re-encrypt to content so as to preserve the original copyright protection.

All receivers operating on the content are likely to have valid decryption keys and may attempt to verify authorization of other peers using decrypted content, but even the unintended possibility of distributing the content without protection to unauthorized peers is likely to cause the original distributors of the content to react negatively to the system.

Even content that is free from encryption has laws preventing its redistribution, which can quickly be used to outlaw the service. While this cannot prevent users

from operating nodes, it would reduce its viability as a high-density peer-to-peer system. The system could be used for a distributor-sanctioned commercially operated system with only producer-provisioned peers providing the repair service and still offer an improvement over the other existing systems, but its complexity compared to DVB-IPTV makes that unlikely.

RELOAD may suffer from a similar problem if the chunks are decrypted, but, as it is already limited to a single time base and requires no canonicalization, it may be able to send the original encrypted chunks without becoming a target for distributors. Our proposed solution could function similarly in a single stream region, but that would negate most of its benefits.

DVB-IPTV bypasses the problem by requiring provisioned infrastructure that is standardized and easily licensed. The receivers solely function as consumers of the service and never redistribute any content.

5. CONCLUSION

This thesis presented a novel solution for the repair of DVB transmissions based on mature technologies. The design still leaves a lot untouched, but should provide a basic outline of one possible system that is fully peer-to-peer while still providing results with a high probability of success.

Our assessment found that our system provided some unique benefits over the two other existing systems that we presented in Section 2.2, but that its complexity may outweight its benefits. Large-scale deployment is also unlikely to be sanctioned by content distributors, as its peer-to-peer nature would involve uncontrolled redistribution of the copyrighted content.

One thing to note when reading this thesis is that the DVB consortium already defines a next-generation standard with improved FEC protection and increased framing. While global deployment may take close to a decade, it may be more efficient to attempt to work for a standardized solution instead of an auxiliary system. Simply the presence of a reliable sequence numbering of packets would solve most of the problems noted in this thesis.

However, this does not mean that the techniques explored here are of no use. There may still exist other infrastructures where the data framing cannot be modified and where data differencing is the only solution for error detection and correction. The ubiquity of bidirectional connectivity between receivers provides us with an excellent platform for distributed error correction.

The techniques may also prove useful for offline operation. Increasingly more data is being stored on shared servers in the cloud. Block deduplication techniques may not be sufficient for handling audiovisual data the comes from multiple sources. The canonicalization and hashing stages presented in this thesis would allow efficient deduplication of large amounts of data as it is being uploaded.

Future work related to this thesis would likely involve a deep study of audio

and video fingerprinting techniques. Perceptual hashing presents a significantly more robust system for identifying audiovisual data, but also makes reconstruction exceedingly difficult without sacrificing quality.

The system could also be extended to better support delayed Video on Demand (VoD). Cheap Digital Video Recorders (DVR) with large amounts of storage could turn the repair service into a distributed VoD service with a long cache of audiovisual content. Such a hybrid system would be able to source itself from the highly bandwidth-efficient DVB network while still providing multi-source and localized VoD. This would not only lessen the load on backbone IP links, but also allow for efficient sharing of resources, without excessive duplication of content.

REFERENCES

[1] Tanenbaum, A. Computer Networks, 4th Edition, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2003. pp. 383-387.

[2] Digita coverage maps for their Finnish DVB-T network [Online] [Referenced 27.8.2012] Available at http://www.digita.fi/kuluttajat/tv/nakyvyysalueet

[3] Mysid. A locator map of Finland [Online] [Referenced 27.8.2012] Available at: http://commons.wikimedia.org/wiki/File:Finland_locator_map.svg

[4] Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for cable systems. ETSI EN 300 429, 1998.

[5] Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services. ETSI EN 300 421, 1997.

[6] Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television. ETSI EN 300 744, 2009.

[7] Information Technology - Generic coding of moving pictures and associated audio information: Systems. ISO/IEC 13818, 2006.

[8] Peterson, L., Davie, B. Computer Networks: A Systems Approach, 3rd Edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003. pp. 97-111.

[9] Digital Video Broadcasting (DVB); Transport of MPEG-2 TS Based DVB Services over IP Based Networks. ETSI TS 102 034, 2009.

[10] Weng, YT., Shieh, CK., Huang, TC., Miao, YB. Using P2P Networks to Repair Packet Losses in Digital Video Broadcasting Systems. Proceedings of the 2011 IEEE 17th International Conferenence on Parallel and Distributed Systems (ICPADS '11), IEEE Computer Society, Washington, DC, USA, 2011.

[11] Information Technology - Coding of audio-visual objects. ISO/IEC 14496, 2005.

[12] Cormen, T., Leiserson, C., Rivest, R. Introduction to Algorithms, MIT press, Cambridge, MA, USA, 1990. pp. 314-320.

[13] Viswanathan, K., Swarminathan, R. Improved String Reconstruction over Insertion-Deletion Channels. Proceedings of the Nineteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '08), Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[14] Cormen, T., Leiserson, C., Rivest, R. Introduction to Algorithms, MIT press, Cambridge, MA, USA, 1990. pp. 226-232.

[15] Zauner, C. Implementation and Benchmarking of Perceptual Image Hash Functions, Master's thesis, Upper Austria University of Applied Sciences, Hagenberg Campus, 2010.

A. TRANSPORT STREAM DIFF

Side-by-side diff of two Transport Streams from different regions. A "|" indicates a conflicting line, "<" a line only present in the left listing and a ">" correspondingly for the right listing.

```
Sync-Byte 0x47: 71 (0x47)                                                  | Sync-Byte 0x47: 71 (0x47)
Transport_error_indicator: 0 (0x00)  [= packet ok]                         | Transport_error_indicator: 0 (0x00)  [= packet ok]
Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]          | Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]
transport_priority: 0 (0x00)                                               | transport_priority: 0 (0x00)
PID: 512 (0x0200)  [= ]                                                    | PID: 512 (0x0200)  [= ]
transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payl | transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payl
adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]   | adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]
continuity_counter: 1 (0x01)  [= (sequence ok)]                            | continuity_counter: 14 (0x0e)  [= (sequence ok)]
    Payload: (len: 184)                                                    |     Payload: (len: 184)
    Data-Bytes:                                                            |     Data-Bytes:
          0000:  db 49 fe 6a a6 6d 5c 40  c2 fa aa 75 75 75 75 75   .I.j.m\@...uuuuu |           0000:  db 49 fe 6a a6 6d 5c 40  c2 fa aa 75 75 75 75 75   .I.j.m\@...uuuuu
          0010:  75 75 75 50 b8 83 7d 53  06 6a a0 00 00 01 0b 32   uuuP..}S.j.....2 |           0010:  75 75 75 50 b8 83 7d 53  06 6a a0 00 00 01 0b 32   uuuP..}S.j.....2
          0020:  ae ae a4 1a 98 37 53 05  2a 42 c5 4c 77 10 6b 55   .....7S.*B.Lw.kU |           0020:  ae ae a4 1a 98 37 53 05  2a 42 c5 4c 77 10 6b 55   .....7S.*B.Lw.kU
          0030:  53 0d 48 10 aa c5 5d 48  75 32 62 73 2f dd 71 02   S.H...]Hu2bs/.q. |           0030:  53 0d 48 10 aa c5 5d 48  75 32 62 73 2f dd 71 02   S.H...]Hu2bs/.q.
          0040:  1b 5d 18 31 a0 61 d0 ae  b9 d5 54 87 53 16 6a 41   .].1.a....T.S.jA |           0040:  1b 5d 18 31 a0 61 d0 ae  b9 d5 54 87 53 16 6a 41   .].1.a....T.S.jA
          0050:  a9 8a 55 20 62 a6 1a 90  fc 60 8a 3d bd e7 b2 13   ..U b....`.=.... |           0050:  a9 8a 55 20 62 a6 1a 90  fc 60 8a 3d bd e7 b2 13   ..U b....`.=....
          0060:  cf 80 4b fa 99 07 6d 2c  d0 1a 1f 23 99 9c 8e 77   ..K...m,...#...w |           0060:  cf 80 4b fa 99 07 6d 2c  d0 1a 1f 23 99 9c 8e 77   ..K...m,...#...w
          0070:  50 92 74 91 5e c0 ee bb  aa 57 ab a9 fe ae ae ae   P.t.^....W...... |           0070:  50 92 74 91 5e c0 ee bb  aa 57 ab a9 fe ae ae ae   P.t.^....W......
          0080:  ae ae ae ae a4 0d 55 dc  ee aa 60 95 56 00 00 00   ......U...`.V... |           0080:  ae ae ae ae a4 0d 55 dc  ee aa 60 95 56 00 00 00   ......U...`.V...
          0090:  01 0c 33 cf d0 aa a4 1b  8c 38 f7 5f ff 8a ba 6c   ..3......8._...l |           0090:  01 0c 33 cf d0 aa a4 1b  8c 38 f7 5f ff 8a ba 6c   ..3......8._...l
          00a0:  82 f3 aa 63 9c b1 af 5c  ef aa 43 ab a9 8e e2 0d   ...c...\..C..... |           00a0:  82 f3 aa 63 9c b1 af 5c  ef aa 43 ab a9 8e e2 0d   ...c...\..C.....
          00b0:  f5 cf 7a 55 09 5d 5d 5d                           ..zU.]]] |           00b0:  f5 cf 7a 55 09 5d 5d 5d                           ..zU.]]]
============================================================               ============================================================

Sync-Byte 0x47: 71 (0x47)                                                  | Sync-Byte 0x47: 71 (0x47)
Transport_error_indicator: 0 (0x00)  [= packet ok]                         | Transport_error_indicator: 0 (0x00)  [= packet ok]
Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]          | Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]
transport_priority: 0 (0x00)                                               | transport_priority: 0 (0x00)
PID: 512 (0x0200)  [= ]                                                    | PID: 512 (0x0200)  [= ]
transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payl | transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payl
adaptation_field_control: 3 (0x03)  [= adaptation_field followed by payload | adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]
continuity_counter: 2 (0x02)  [= (sequence ok)]                            | continuity_counter: 15 (0x0f)  [= (sequence ok)]
    Adaptation_field:                                                      |     Payload: (len: 184)
        adaptation_field_length: 7 (0x07)                                  <
        discontinuity_indicator: 0 (0x00)                                  <
        random_access_indicator: 0 (0x00)                                  <
        elementary_stream_priotity_indicator: 0 (0x00)                     <
        PCR_flag: 1 (0x01)                                                 <
        OPCR_flag: 0 (0x00)                                                <
        splicing_point_flag: 0 (0x00)                                      <
        transport_private_data_flag: 0 (0x00)                              <
        adaptation_field_extension_flag: 0 (0x00)                          <
        program_clock_reference:                                           <
            baseH: 1 (0x01)                                                <
            baseL: 4140090535 (0xf6c4c4a7)                                 <
            reserved: 63 (0x3f)                                            <
            extension: 95 (0x005f)                                         <
             ==> program_clock_reference: 2530517349395 (0x24d2e967413)  [= PCR-Times <
    Payload: (len: 176)                                                    <
    Data-Bytes:                                                            |     Data-Bytes:
          0000:  4c 35 77 3d 7a a4 1e 9f  31 fd ea 90 ea ea 61 a9   L5w=z...1.....a. |           0000:  4c 35 77 3d 7a a4 1e 9f  31 fd ea 90 ea ea 61 a9   L5w=z...1.....a.
          0010:  07 cd 1d c0 50 06 0a 28  b4 02 28 02 06 f2 87 28   ....P..(..(....( |           0010:  07 cd 1d c0 50 06 0a 28  b4 02 28 02 06 f2 87 28   ....P..(..(....(
          0020:  fd 13 09 a8 db 01 93 cb  2b 67 16 30 c8 79 3e 9c   ........+g.0.y>. |           0020:  fd 13 09 a8 db 01 93 cb  2b 67 16 30 c8 79 3e 9c   ........+g.0.y>.
          0030:  63 52 39 10 e0 f8 03 b7  23 4c 6b c4 4a bc 43 f8   cR9.....#Lk.J.C. |           0030:  63 52 39 10 e0 f8 03 b7  23 4c 6b c4 4a bc 43 f8   cR9.....#Lk.J.C.
          0040:  19 0d 4f 2c 0c f5 8f 11  06 c3 5f 3c 92 88 9e f1   ..O,......_<.... |           0040:  19 0d 4f 2c 0c f5 8f 11  06 c3 5f 3c 92 88 9e f1   ..O,......_<....
          0050:  8d 85 41 b8 f8 b4 b3 3f  39 c8 0f 0f d1 34 09 6e   ..A....?9....4.n |           0050:  8d 85 41 b8 f8 b4 b3 3f  39 c8 0f 0f d1 34 09 6e   ..A....?9....4.n
          0060:  39 24 61 9a e9 ce 03 b0  88 a4 04 7e 8a a6 5a ba   9$a........~..Z. |           0060:  39 24 61 9a e9 ce 03 b0  88 a4 04 7e 8a a6 5a ba   9$a........~..Z.
          0070:  ba ba ba ba a8 54 c3 52  0d c6 2c 16 ee 9c 45 de   .....T.R..,...E. |           0070:  ba ba ba ba a8 54 c3 52  0d c6 2c 16 ee 9c 45 de   .....T.R..,...E.
          0080:  b8 86 ba b8 81 8b aa 98  20 00 00 01 0d 32 ae e7   ........ ....2.. |           0080:  b8 86 ba b8 81 8b aa 98  20 00 00 01 0d 32 ae e7   ........ ....2..
          0090:  bd e7 9d dd 57 d7 ff 9e  76 ba af 5f 57 73 be ab   ....W...v..._Ws.. |           0090:  bd e7 9d dd 57 d7 ff 9e  76 ba af 5f 57 73 be ab   ....W...v..._Ws..
          00a0:  ab d7 eb fa b1 ff ef aa  45 a9 8e ae e3 06 2b aa   ........E.....+. |           00a0:  ab d7 eb fa b1 ff ef aa  45 a9 8e ae e3 06 2b aa   ........E.....+.
                                                                     >           00b0:  a7 72 d4 5e 6a a4 1a ba                           .r.^j...
============================================================               ============================================================

Sync-Byte 0x47: 71 (0x47)                                                  | Sync-Byte 0x47: 71 (0x47)
Transport_error_indicator: 0 (0x00)  [= packet ok]                         | Transport_error_indicator: 0 (0x00)  [= packet ok]
Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]          | Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]
transport_priority: 0 (0x00)                                               | transport_priority: 0 (0x00)
PID: 512 (0x0200)  [= ]                                                    | PID: 512 (0x0200)  [= ]
transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payl | transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payl
adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]   | adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]
continuity_counter: 3 (0x03)  [= (sequence ok)]                            | continuity_counter: 0 (0x00)  [= (sequence ok)]
    Payload: (len: 184)                                                    |     Payload: (len: 184)
    Data-Bytes:                                                            |     Data-Bytes:
```

```
0000:  a7 72 d4 5e 6a a4 1a ba  98 b1 52 04 2a 60 c5 48   .r.^j.....R.*`.H  |
0010:  10 eb d8 c9 51 a5 44 65  cf dc 71 1e 08 09 cc 3c   ....Q.De..q....<  |
0020:  e1 27 1f 77 46 13 09 8b  66 0d 72 f6 25 84 23 3f   .'.wF...f.r.%.#?  |
0030:  f0 68 0e cb 00 a9 20 61  0f 8e 58 c3 c9 fc eb 1f   .h.... a..X.....  |
0040:  92 7c 92 5f c7 ec 21 5e  9c 41 50 6e 28 99 f0 17   .|._..!^.APn(...  |
0050:  02 e2 96 89 3b 66 56 19  b3 41 29 ee e2 26 fd 02   ....;fV..A)..&..  |
0060:  ea f3 a8 a6 3a 73 b4 7b  5c d7 9f 00 27 c4 11 7d   ....:s.{\...'..}  |
0070:  9e 46 0f 75 ae 9c 41 ce  d2 0b 36 2a ea ea fc ec   .F.u..A...6*....  |
0080:  cf ab 5d 1a 98 31 52 04  2e 20 42 aa ab 15 30 eb   ..]..1R.. B...0.  |
0090:  ea c0 00 00 01 0e 32 a4  28 54 c5 8a ba 98 31 52   ......2.(T....1R  |
00a0:  14 2a ea 61 a9 03 17 18  a3 9d 73 8f 74 e7 4c aa   .*.a......s.t.L.  |
00b0:  a4 0c 55 d5 dc 41 4f 54                            ..U..AOT          |
============================================================
```

```
Sync-Byte 0x47: 71 (0x47)
Transport_error_indicator: 0 (0x00)  [= packet ok]
Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]
transport_priority: 0 (0x00)
PID: 512 (0x0200)  [= ]
transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payload]
adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]
continuity_counter: 4 (0x04)  [= (sequence ok)]
    Payload: (len: 184)
    Data-Bytes:
          0000:  c3 57 e7 dd 55 d5 eb da  bb 9c 3b aa 60 85 48 18   .W..U.....;.`.H.  |
          0010:  e8 c7 63 23 d1 c6 70 01  f9 18 74 9c a1 39 f8 c0   ..c#..p...t..9..  |
          0020:  fb bc 41 b1 c9 af 89 38  76 0a 10 65 e6 36 6e 60   ..A.....8v..e.6n`  |
          0030:  88 e6 bc 63 66 82 9c 9a  59 94 7c 9f 6d 9e ee a8   ...cf...Y.|.m...  |
          0040:  3a 04 7a 3f f7 70 e9 26  eb 39 85 49 c0 6c 46 3f   :.z?.p.&.9.I.lF?  |
          0050:  dd 7a 06 3f e0 3d 0a 0a  b3 ab 1c 24 8c 9b c4 97   .z.?.=.....$....  |
          0060:  01 a6 9f 10 58 67 dd e7  c7 ba a6 5a ba ba ba bf   ....Xg.....Z....  |
          0070:  10 7b ef 3d 3d 57 73 de  ab ab 00 00 01 0f 32 a4   .{.==Ws.......2.  |
          0080:  19 cd 69 bc d8 74 da ab  15 50 aa cd 54 6a 60 85   ..i..t...P..Tj`.  |
          0090:  cb 35 d5 20 42 ae a4 1a  98 6a 61 f3 41 de ab ab   .5. B....ja.A...  |
          00a0:  ab ab ab f3 f3 aa 60 87  4d 90 48 2b ea bf 34 09   ......`.M.H+..4.  |
          00b0:  2f bc 63 44 d4 fb bc f3                            /.cD....          |
============================================================
```

```
Sync-Byte 0x47: 71 (0x47)
Transport_error_indicator: 0 (0x00)  [= packet ok]
Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]
transport_priority: 0 (0x00)
PID: 512 (0x0200)  [= ]
transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payload]
adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]
continuity_counter: 5 (0x05)  [= (sequence ok)]
    Payload: (len: 184)
    Data-Bytes:
          0000:  39 38 71 a3 19 85 9b 07  8e 70 eb 02 61 60 10 6c   98q.......p..a`.l  |
          0010:  e1 77 77 96 39 39 d9 84  dd e2 0b 0f f5 cd 26 ea   .ww.99........&.  |
          0020:  81 8b 04 08 24 17 bf 66  c3 dc 2b 92 78 f3 5a 51   ....$..f..+.x.ZQ  |
          0030:  d5 8f 7c 71 13 de 31 52  c1 5e ee 9f d0 d0 bc de   ..|q..1R.^......  |
          0040:  16 1f af 3d f9 81 f1 84  f8 78 eb cd 85 c9 cb 87   ...=.....x......  |
          0050:  70 08 6e f1 05 84 20 ce  e2 76 b5 46 5c d4 c6 d5   p.n... ..v.F\...  |
          0060:  d5 d5 be 20 f9 4f 5e c1  86 61 4f 00 00 00 01 10   ... .O^..aO.....  |
          0070:  32 ae 72 d2 e7 bc 60 8f  2e ea ba ac 54 c3 73 ae   2.r...`.....T.s.  |
          0080:  b9 dd 4a a1 73 ae a9 02  15 20 dc c6 3b bb c4 16   ..J.s.... ..;...  |
          0090:  14 f5 48 77 18 c5 3b ba  a6 08 79 d8 be a9 06 a4   ..Hw..;...y.....  |
          00a0:  0c 5c 62 8e f5 48 18 f3  ef bc f5 f7 8c 39 dc 9d   .\b..H.......9..  |
          00b0:  86 fc 87 3c fb c6 14 7f                            ...<....          |
============================================================
```

```
Sync-Byte 0x47: 71 (0x47)
Transport_error_indicator: 0 (0x00)  [= packet ok]
Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]
transport_priority: 0 (0x00)
PID: 512 (0x0200)  [= ]
transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payload]
| adaptation_field_control: 3 (0x03)  [= adaptation_field followed by payload]
| continuity_counter: 1 (0x01)  [= (sequence ok)]
|     Adaptation_field:
>         adaptation_field_length: 7 (0x07)
>         discontinuity_indicator: 0 (0x00)
>         random_access_indicator: 0 (0x00)
>         elementary_stream_priotity_indicator: 0 (0x00)
>         PCR_flag: 1 (0x01)
>         OPCR_flag: 0 (0x00)
>         splicing_point_flag: 0 (0x00)
>         transport_private_data_flag: 0 (0x00)
>         adaptation_field_extension_flag: 0 (0x00)
>         program_clock_reference:
>             baseH: 0 (0x00)
>             baseL: 2040854878 (0x79a4f95e)
>             reserved: 63 (0x3f)
>             extension: 90 (0x005a)
>              ==> program_clock_reference: 612256463490 (0x8e8d543a82)  [= PCR-Timesta
>     Payload: (len: 176)
      Data-Bytes:
          0000:  bb 9c 3b aa 60 85 48 18  e8 c7 63 23 d1 c6 70 01   ..;.`.H...c#..p.  |
          0010:  f9 18 74 9c a1 39 f8 c0  fb bc 41 b1 c9 af 89 38   ..t..9....A....8  |
          0020:  76 0a 10 65 e6 36 6e 60  88 e6 bc 63 66 82 9c 9a   v..e.6n`...cf...  |
          0030:  59 94 7c 9f 6d 9e ee a8  3a 04 7a 3f f7 70 e9 26   Y.|.m...:.z?.p.&  |
          0040:  eb 39 85 49 c0 6c 46 3f  dd 7a 06 3f e0 3d 0a 0a   .9.I.lF?.z.?.=..  |
          0050:  b3 ab 1c 24 8c 9b c4 97  01 a6 9f 10 58 67 dd e7   ...$........Xg..  |
          0060:  c7 ba a6 5a ba ba ba bf  10 7b ef 3d 3d 57 73 de   ...Z.....{.==Ws.  |
          0070:  ab ab 00 00 01 0f 32 a4  19 cd 69 bc d8 74 da ab   ......2...i..t..  |
          0080:  15 50 aa cd 54 6a 60 85  cb 35 d5 20 42 ae a4 1a   .P..Tj`..5. B...  |
          0090:  98 6a 61 f3 41 de ab ab  ab ab f3 f3 aa 60 87      .ja.A.........`.  |
          00a0:  4d 90 48 2b ea bf 34 09  2f bc 63 44 d4 fb bc f3   M.H+..4./.cD....  |
          00b0:                                                     <
<============================================================
```

```
Sync-Byte 0x47: 71 (0x47)
Transport_error_indicator: 0 (0x00)  [= packet ok]
Payload_unit_start_indicator: 0 (0x00)  [= Packet data continues]
transport_priority: 0 (0x00)
PID: 512 (0x0200)  [= ]
transport_scrambling_control: 0 (0x00)  [= No scrambling of TS packet payload]
adaptation_field_control: 1 (0x01)  [= no adaptation_field, payload only]
| continuity_counter: 2 (0x02)  [= (sequence ok)]
    Payload: (len: 184)
    Data-Bytes:
          0000:  39 38 71 a3 19 85 9b 07  8e 70 eb 02 61 60 10 6c   98q.......p..a`.l  |
          0010:  e1 77 77 96 39 39 d9 84  dd e2 0b 0f f5 cd 26 ea   .ww.99........&.  |
          0020:  81 8b 04 08 24 17 bf 66  c3 dc 2b 92 78 f3 5a 51   ....$..f..+.x.ZQ  |
          0030:  d5 8f 7c 71 13 de 31 52  c1 5e ee 9f d0 d0 bc de   ..|q..1R.^......  |
          0040:  16 1f af 3d f9 81 f1 84  f8 78 eb cd 85 c9 cb 87   ...=.....x......  |
          0050:  70 08 6e f1 05 84 20 ce  e2 76 b5 46 5c d4 c6 d5   p.n... ..v.F\...  |
          0060:  d5 d5 be 20 f9 4f 5e c1  86 61 4f 00 00 00 01 10   ... .O^..aO.....  |
          0070:  32 ae 72 d2 e7 bc 60 8f  2e ea ba ac 54 c3 73 ae   2.r...`.....T.s.  |
          0080:  b9 dd 4a a1 73 ae a9 02  15 20 dc c6 3b bb c4 16   ..J.s.... ..;...  |
          0090:  14 f5 48 77 18 c5 3b ba  a6 08 79 d8 be a9 06 a4   ..Hw..;...y.....  |
          00a0:  0c 5c 62 8e f5 48 18 f3  ef bc f5 f7 8c 39 dc 9d   .\b..H.......9..  |
          00b0:  86 fc 87 3c fb c6 14 7f                            ...<....          |
============================================================
```

B. PACKETIZED ELEMENTARY STREAM VIDEO DIFF

Side-by-side diff of two video PES from different regions. A ‘|” indicates a conflicting line, ‘<” a line only present in the left listing and a ‘>” correspondingly for the right listing.

```
TS sub-decoding (103 packet(s) stored for PID 0x0202):
======================================================
TS contains PES/PS stream...
PS/PES packet (length=19):
    Packet_start_code_prefix: 0x000001
    Stream_id: 224 (0xe0)  [= ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172
    PES_packet_length: 0 (0x0000)
     ==> unbound video elementary stream...

        reserved1: 2 (0x02)
        PES_scrambling_control: 0 (0x00)  [= not scrambled]
        PES_priority: 0 (0x00)
        data_alignment_indicator: 1 (0x01)
        copyright: 0 (0x00)
        original_or_copy: 0 (0x00)
        PTS_DTS_flags: 3 (0x03)
        ES_rate_flag: 0 (0x00)
        additional_copy_info_flag: 0 (0x00)
        PES_CRC_flag: 0 (0x00)
        PES_extension_flag: 0 (0x00)
        PES_header_data_length: 10 (0x0a)
        PTS:
           Fixed: 3 (0x03)
           PTS:
              bit[32..30]: 1 (0x01)
              marker_bit: 1 (0x01)
              bit[29..15]: 14127 (0x372f)                                  |
              marker_bit: 1 (0x01)
              bit[14..0]: 6528 (0x1980)                                    |
              marker_bit: 1 (0x01)
               ==> PTS: 1536661888 (0x5b979980)  [= 90 kHz-Timestamp: 4:44:34.020 |
        DTS:
           Fixed: 1 (0x01)
           DTS:
              bit[32..30]: 1 (0x01)
              marker_bit: 1 (0x01)
              bit[29..15]: 14126 (0x372e)                                  |
              marker_bit: 1 (0x01)
              bit[14..0]: 28496 (0x6f50)                                   |
              marker_bit: 1 (0x01)
               ==> DTS: 1536651088 (0x5b976f50)  [= 90 kHz-Timestamp: 4:44:33.900 |

PS/PES packet (length=9):
    Packet_start_code_prefix: 0x000001
    Stream_id: 0 (0x00)  [= picture_start_code]
        temporal_reference: 5 (0x0005)
        picture_coding_type: 2 (0x02)  [= predictive-coded (P)]
        vbv_delay: 65535 (0xffff)
        full_pel_forward_vector: 1 (0x01)
        forward_f_code: 7 (0x07)
        extra_bit_picture: 1 (0x01)
        extra_information_picture: 255 (0xff)
        extra_bit_picture: 0 (0x00)

PS/PES packet (length=9):                                                  |
    Packet_start_code_prefix: 0x000001
    Stream_id: 181 (0xb5)  [= extension_start_code]
        extension_start_code_identifier: 8 (0x08)  [= Picture Coding Extension ID
        f_code[forward][horizontal]: 5 (0x05)
        f_code[forward][vertical]: 4 (0x04)
        f_code[backward][horizontal]: 15 (0x0f)
        f_code[backward][vertical]: 15 (0x0f)
        intra_dc_precision: 0 (0x00)  [= 8 bits]
        picture_structure: 3 (0x03)  [= frame picture]
        top_field_first: 1 (0x01)
        frame_pred_frame_dct: 0 (0x00)
        concealment_motion_vectors: 0 (0x00)
        q_scale_type: 1 (0x01)
        intra_vlc_format: 1 (0x01)
        alternate_scan: 0 (0x00)
```

```
TS sub-decoding (103 packet(s) stored for PID 0x0202):
======================================================
TS contains PES/PS stream...
PS/PES packet (length=19):
    Packet_start_code_prefix: 0x000001
    Stream_id: 224 (0xe0)  [= ITU-T Rec. H.262 | ISO/IEC 13818-2 or ISO/IEC 11172
    PES_packet_length: 0 (0x0000)
     ==> unbound video elementary stream...

        reserved1: 2 (0x02)
        PES_scrambling_control: 0 (0x00)  [= not scrambled]
        PES_priority: 0 (0x00)
        data_alignment_indicator: 1 (0x01)
        copyright: 0 (0x00)
        original_or_copy: 0 (0x00)
        PTS_DTS_flags: 3 (0x03)
        ES_rate_flag: 0 (0x00)
        additional_copy_info_flag: 0 (0x00)
        PES_CRC_flag: 0 (0x00)
        PES_extension_flag: 0 (0x00)
        PES_header_data_length: 10 (0x0a)
        PTS:
           Fixed: 3 (0x03)
           PTS:
              bit[32..30]: 1 (0x01)
              marker_bit: 1 (0x01)
              bit[29..15]: 600 (0x0258)
              marker_bit: 1 (0x01)
              bit[14..0]: 2104 (0x0838)
              marker_bit: 1 (0x01)
               ==> PTS: 1093404728 (0x412c0838)  [= 90 kHz-Timestamp: 3:22:28.941
        DTS:
           Fixed: 1 (0x01)
           DTS:
              bit[32..30]: 1 (0x01)
              marker_bit: 1 (0x01)
              bit[29..15]: 599 (0x0257)
              marker_bit: 1 (0x01)
              bit[14..0]: 24072 (0x5e08)
              marker_bit: 1 (0x01)
               ==> DTS: 1093393928 (0x412bde08)  [= 90 kHz-Timestamp: 3:22:28.821

PS/PES packet (length=9):
    Packet_start_code_prefix: 0x000001
    Stream_id: 0 (0x00)  [= picture_start_code]
        temporal_reference: 5 (0x0005)
        picture_coding_type: 2 (0x02)  [= predictive-coded (P)]
        vbv_delay: 65535 (0xffff)
        full_pel_forward_vector: 1 (0x01)
        forward_f_code: 7 (0x07)
        extra_bit_picture: 1 (0x01)
        extra_information_picture: 255 (0xff)
        extra_bit_picture: 0 (0x00)

PS/PES packet (length=11):
    Packet_start_code_prefix: 0x000001
    Stream_id: 181 (0xb5)  [= extension_start_code]
        extension_start_code_identifier: 8 (0x08)  [= Picture Coding Extension ID
        f_code[forward][horizontal]: 5 (0x05)
        f_code[forward][vertical]: 4 (0x04)
        f_code[backward][horizontal]: 15 (0x0f)
        f_code[backward][vertical]: 15 (0x0f)
        intra_dc_precision: 0 (0x00)  [= 8 bits]
        picture_structure: 3 (0x03)  [= frame picture]
        top_field_first: 1 (0x01)
        frame_pred_frame_dct: 0 (0x00)
        concealment_motion_vectors: 0 (0x00)
        q_scale_type: 1 (0x01)
        intra_vlc_format: 1 (0x01)
        alternate_scan: 0 (0x00)
```

```
    repeat_first_field: 0 (0x00)
    chroma_420_type: 0 (0x00)
    progressive_frame: 0 (0x00)
    composite_display_flag: 0 (0x00)

PS/PES packet (length=387):                                                    |
  Packet_start_code_prefix: 0x000001
  Stream_id: 1 (0x01)  [= slice_start_code]
    MPEG-2 Slice (incl. sync + id):
      0000:  00 00 01 01 23 9f 72 24  1e a8 cf e5 18 f1 e2 04   ....#.r$..
      0010:  0c 7c ea b1 46 d1 a1 d6  70 8b 8e 88 6c 2f c2 87   .|..F...p.
      0020:  96 70 1e 53 37 92 53 fd  11 93 71 4a b5 06 c6 60   .p.S7.S...
      0030:  e0 9b 0f 69 ac 86 b4 69  42 cf 66 ae 09 08 46 14   ...i...iB.
      0040:  fb 62 da 72 32 c5 d8 71  4c 23 11 98 d9 6c 2d 43   .b.r2..qL#
      0050:  02 7d 4f 8c 69 92 9e cb  4a a3 61 fe 0c 5f 66 1e   .}O.i...J.
      0060:  77 ab 43 4a 34 62 7b 42  ec 2a 2d 1c ad 0d 0d 2e   w.CJ4b{B.*
      0070:  8f 68 55 2c 35 86 23 46  85 96 55 08 02 da 51 a1   .hU,5.#F..
      0080:  a1 ac 15 29 65 34 1f 0f  66 c3 1e 52 47 2f 69 6b   ...)e4..f.
      0090:  6a e6 0f 1e 3a e8 d6 19  cf 74 3b 10 05 65 30 71   j...:....t
      00a0:  1a 4b 6e e8 d1 81 de b1  b9 82 c0 1f 47 2d 8d a7   .Kn.....
      00b0:  97 19 03 e2 34 59 63 4c  90 f6 8d 75 c5 a4 2b 69   ....4YcL..
      00c0:  0d 5b 37 b7 f9 db 9e 7c  8b 8e ca 87 5b 68 f6 d5   .[7....|..
      00d0:  c9 1f 8c b6 60 7d 0c 2a  35 5f e0 07 80 0e c0 19   ....`}.*5_
      00e0:  d7 e0 85 c7 44 36 95 22  d1 8a 7a d6 66 5d 29 92   ....D6."..
      00f0:  56 e5 23 08 78 b0 d4 37  e1 a1 bc e0 3e 5a 0e 0f   V.#.x..7..
      0100:  a0 d0 dd 8f ee ed dd 2e  60 d6 51 e6 9d e4 6e bf   ........`.
      0110:  98 61 78 fc ab 24 67 37  1f 26 71 e6 89 b9 98 18   .ax..$g7.&
      0120:  18 06 00 48 31 66 ec c7  40 3b 28 9a 03 b2 62 18   ...H1f..@;
      0130:  a0 1d 81 50 c4 2c 20 a2  19 61 99 c7 81 6e 88 a4   ...P., ..a
      0140:  02 1f fd e6 04 4f fc c4  fc 69 fc 78 e3 22 d9 09   .....O...i
      0150:  40 17 71 37 6f 73 e5 67  e9 2c 96 e9 49 e3 7f 7e   @.q7os.g.,
      0160:  91 7d 7f 85 ae d5 cd ad  b1 2f b2 dc ac 03 31 63   .}......./
      0170:  85 77 9c d6 32 0a ca 00  ef cf 0f d0 54 93 f0 7c   .w..2.....
      0180:  7c 16 6c                                          |.l        |

PS/PES packet (length=407):                                                    |
  Packet_start_code_prefix: 0x000001
  Stream_id: 2 (0x02)  [= slice_start_code]
    MPEG-2 Slice (incl. sync + id):
      0000:  00 00 01 02 23 98 12 67  0e 65 43 9e 7e 64 6c a8   ....#..g.e
      0010:  c6 e4 a5 30 b2 d0 ea c6  0e a7 a3 09 02 e3 14 b9   ...0......
      0020:  42 fd 1b 36 1a 34 d2 a0  c3 86 0c 68 56 a5 2c 6c   B..6.4.....
      0030:  02 eb 04 0c 28 6b 42 ad  1c 3d 81 ac 86 b4 67 89   ....(kB..=
      0040:  0a 8c 16 2b 2a 29 de 8d  29 67 33 1a cc 6b 31 b9   ...+*)..)g
      0050:  b3 24 94 b5 2e d6 3b 06  06 23 2e ab 1d 94 b3 e9   .$....;..#
      0060:  6d 08 c5 68 cd b9 10 b1  87 b3 2c 94 36 1f 19 0a   m..h......
      0070:  09 28 b6 6b 14 b1 cc e3  15 a3 4a 79 ca ad 10 b3   .(.k......
      0080:  8e 1a a8 8c 12 a1 51 8d  28 ec 8b 19 53 d9 2b 46   ......Q.(.
      0090:  88 69 e3 51 a4 54 1b 6c  a3 46 b4 5a aa 85 c1 90   .i.Q.T.l.F
      00a0:  a6 1b 02 65 66 31 88 d1  9a 9f 6b 3d 0d 92 86 9e   ...ef1....
      00b0:  c8 51 6a 78 d8 cc 91 2a  24 61 86 8c 74 a8 30 a0   .Qjx...*$a
      00c0:  c6 32 4a 7d 85 b0 be 16  34 a5 d0 d4 61 59 46 8c   .2J}....4.
      00d0:  7c a2 a9 ec 21 70 e6 86  8e 62 34 f6 48 53 3a cb   |...!p...b
      00e0:  10 20 32 9b 19 0e b9 41  5e e1 2b c0 3d 3c 8e b1   . 2....A^.
      00f0:  f7 cd a1 a3 0c 3f 1e b7  93 f7 c2 38 8b 73 ee 30   .....?....
      0100:  40 c9 41 4b 61 f1 41 3f  1e 4a 40 f6 70 e6 77 3c   @.AKa.A?.J
      0110:  2e 53 8c c2 d7 c3 de 3d  7c 7a 96 66 69 fc 82 0d   .S.....=|z
      0120:  cc c2 66 1a 6b 27 2f 0e  78 35 8c c7 1d b7 1e aa   ..f.k'/.x5
      0130:  61 f6 b1 07 01 19 95 01  cb 9f 29 92 34 a4 b0 c3   a.........
      0140:  c6 ef c9 e7 ad 9b 18 15  7f 89 1d 84 39 92 a5 d3   ........
      0150:  7c cc 82 5d f3 1c dc ce  3c 8b a5 fc af b2 04 7d   |..]....<.
      0160:  e7 43 94 ca 5a 9d cc 57  1f 61 3f ba b3 f0 27 b4   .C..Z..W.a
      0170:  9e 74 e6 5c d5 12 d0 3e  5b 9b 3f 1e 4f 3a 52 3a   .t.\...>[.
      0180:  fe 58 f7 f3 5c f8 52 ee  48 08 fe 01 13 e2 3d 37   .X..\.R.H.
      0190:  67 3c 2b 53 60 a9 b0                               g<+S`..    |

PS/PES packet (length=291):                                                    |
  Packet_start_code_prefix: 0x000001
  Stream_id: 3 (0x03)  [= slice_start_code]
    MPEG-2 Slice (incl. sync + id):
      0000:  00 00 01 03 23 9a 9f 9c  17 29 ce 94 a0 bd 33 2e   ....#....)
      0010:  69 8d c8 a5 5a c7 56 0e  61 55 a0 f6 86 6c 10 36   i...Z.V.aU
      0020:  6a b2 31 ba 87 46 19 c7  3d 9d 9e 28 53 52 ef f4   j.1..F..=.
      0030:  c2 fb d0 99 0a 21 34 dd  91 92 56 8d a2 5b a5 42   .....!4...
      0040:  c0 3f 8d 53 be 9c e8 0e  69 93 d5 aa de fa 11 13   .?.S....i.
      0050:  8c 91 ae e4 8a fd 1a 8a  d5 bc 9e 8d 33 8a ad c4   ........
      0060:  48 29 a5 bb ca e7 6c ff  c4 d0 1b 64 00 4e 58 71   H)....l...
      0070:  fe 4f 17 73 30 01 e0 06  b8 68 14 2d 03 f3 c4 31   .O.s0....h
      0080:  a0 37 26 01 44 ba f9 33  0d 15 e4 ba 19 45 36 0b   .7&.D..3..
      0090:  fe 99 85 65 34 8e c6 49  7b 8c 51 33 ec e7 9d 63   ...e4..I{.
      00a0:  72 ff f9 b3 98 38 ff eb  85 79 64 ea 80 ed 37 37   r....8...y
      00b0:  c6 15 7b 19 cf df b0 10  79 22 54 c2 ae b5 19 73   ..{.....y"
      00c0:  31 c7 8b 93 82 57 8e 67  00 8f 10 55 6e ff 3f fe   1....W.g..
      00d0:  52 d2 b3 d8 c3 33 87 9b  66 5e d4 dc dc df 33 26   R....3..f^
      00e0:  47 3f b2 cc 59 16 4e 0e  0f 55 b1 d4 e2 9c 4d db   G?..Y.N..U
      00f0:  aa e6 04 3f 38 d2 3e 69  83 a0 3d 92 b3 03 84 1f   ...?8.>i..
      0100:  a4 d1 05 50 71 bc 36 a9  ec c8 aa c9 62 33 59 5b   ...Pq.6...
      0110:  2a d7 23 31 bc 3d 4b 18  a1 6c aa b1 a1 f5 b2 18   *.#1.=K..l
      0120:  d6 c3 36                                          ..6        |
```

```
    repeat_first_field: 0 (0x00)
    chroma_420_type: 0 (0x00)
    progressive_frame: 0 (0x00)
    composite_display_flag: 0 (0x00)

PS/PES packet (length=388):
  Packet_start_code_prefix: 0x000001
  Stream_id: 1 (0x01)  [= slice_start_code]
    MPEG-2 Slice (incl. sync + id):
      0000:  00 00 01 01 23 9f 72 24  1e a8 cf e5 18 f1 e2 04   ....#.r$..
      0010:  0c 7c ea b1 46 d1 a1 d6  70 8b 8e 88 6c 2f c2 87   .|..F...p.
      0020:  96 70 1e 53 37 92 53 fd  11 93 71 4a b5 06 c6 60   .p.S7.S...
      0030:  e0 9b 0f 69 ac 86 b4 69  42 cf 66 ae 09 08 46 14   ...i...iB.
      0040:  fb 62 da 72 32 c5 d8 71  4c 23 11 98 d9 6c 2d 43   .b.r2..qL#
      0050:  02 7d 4f 8c 69 92 9e cb  4a a3 61 fe 0c 5f 66 1e   .}O.i...J.
      0060:  77 ab 43 4a 34 62 7b 42  ec 2a 2d 1c ad 0d 0d 2e   w.CJ4b{B.*
      0070:  8f 68 55 2c 35 86 23 46  85 96 55 08 02 da 51 a1   .hU,5.#F..
      0080:  a1 ac 15 29 65 34 1f 0f  66 c3 1e 52 47 2f 69 6b   ...)e4..f.
      0090:  6a e6 0f 1e 3a e8 d6 19  cf 74 3b 10 05 65 30 71   j...:....t
      00a0:  1a 4b 6e e8 d1 81 de b1  b9 82 c0 1f 47 2d 8d a7   .Kn.....
      00b0:  97 19 03 e2 34 59 63 4c  90 f6 8d 75 c5 a4 2b 69   ....4YcL..
      00c0:  0d 5b 37 b7 f9 db 9e 7c  8b 8e ca 87 5b 68 f6 d5   .[7....|..
      00d0:  c9 1f 8c b6 60 7d 0c 2a  35 5f e0 07 80 0e c0 19   ....`}.*5_
      00e0:  d7 e0 85 c7 44 36 95 22  d1 8a 7a d6 66 5d 29 92   ....D6."..
      00f0:  56 e5 23 08 78 b0 d4 37  e1 a1 bc e0 3e 5a 0e 0f   V.#.x..7..
      0100:  a0 d0 dd 8f ee ed dd 2e  60 d6 51 e6 9d e4 6e bf   ........`.
      0110:  98 61 78 fc ab 24 67 37  1f 26 71 e6 89 b9 98 18   .ax..$g7.&
      0120:  18 06 00 48 31 66 ec c7  40 3b 28 9a 03 b2 62 18   ...H1f..@;
      0130:  a0 1d 81 50 c4 2c 20 a2  19 61 99 c7 81 6e 88 a4   ...P., ..a
      0140:  02 1f fd e6 04 4f fc c4  fc 69 fc 78 e3 22 d9 09   .....O...i
      0150:  40 17 71 37 6f 73 e5 67  e9 2c 96 e9 49 e3 7f 7e   @.q7os.g.,
      0160:  91 7d 7f 85 ae d5 cd ad  b1 2f b2 dc ac 03 31 63   .}......./
      0170:  85 77 9c d6 32 0a ca 00  ef cf 0f d0 54 93 f0 7c   .w..2.....
      0180:  7c 16 6c 00                                       |.l.

PS/PES packet (length=408):
  Packet_start_code_prefix: 0x000001
  Stream_id: 2 (0x02)  [= slice_start_code]
    MPEG-2 Slice (incl. sync + id):
      0000:  00 00 01 02 23 98 12 67  0e 65 43 9e 7e 64 6c a8   ....#..g.e
      0010:  c6 e4 a5 30 b2 d0 ea c6  0e a7 a3 09 02 e3 14 b9   ...0......
      0020:  42 fd 1b 36 1a 34 d2 a0  c3 86 0c 68 56 a5 2c 6c   B..6.4.....
      0030:  02 eb 04 0c 28 6b 42 ad  1c 3d 81 ac 86 b4 67 89   ....(kB..=
      0040:  0a 8c 16 2b 2a 29 de 8d  29 67 33 1a cc 6b 31 b9   ...+*)..)g
      0050:  b3 24 94 b5 2e d6 3b 06  06 23 2e ab 1d 94 b3 e9   .$....;..#
      0060:  6d 08 c5 68 cd b9 10 b1  87 b3 2c 94 36 1f 19 0a   m..h......
      0070:  09 28 b6 6b 14 b1 cc e3  15 a3 4a 79 ca ad 10 b3   .(.k......
      0080:  8e 1a a8 8c 12 a1 51 8d  28 ec 8b 19 53 d9 2b 46   ......Q.(.
      0090:  88 69 e3 51 a4 54 1b 6c  a3 46 b4 5a aa 85 c1 90   .i.Q.T.l.F
      00a0:  a6 1b 02 65 66 31 88 d1  9a 9f 6b 3d 0d 92 86 9e   ...ef1....
      00b0:  c8 51 6a 78 d8 cc 91 2a  24 61 86 8c 74 a8 30 a0   .Qjx...*$a
      00c0:  c6 32 4a 7d 85 b0 be 16  34 a5 d0 d4 61 59 46 8c   .2J}....4.
      00d0:  7c a2 a9 ec 21 70 e6 86  8e 62 34 f6 48 53 3a cb   |...!p...b
      00e0:  10 20 32 9b 19 0e b9 41  5e e1 2b c0 3d 3c 8e b1   . 2....A^.
      00f0:  f7 cd a1 a3 0c 3f 1e b7  93 f7 c2 38 8b 73 ee 30   .....?....
      0100:  40 c9 41 4b 61 f1 41 3f  1e 4a 40 f6 70 e6 77 3c   @.AKa.A?.J
      0110:  2e 53 8c c2 d7 c3 de 3d  7c 7a 96 66 69 fc 82 0d   .S.....=|z
      0120:  cc c2 66 1a 6b 27 2f 0e  78 35 8c c7 1d b7 1e aa   ..f.k'/.x5
      0130:  61 f6 b1 07 01 19 95 01  cb 9f 29 92 34 a4 b0 c3   a.........
      0140:  c6 ef c9 e7 ad 9b 18 15  7f 89 1d 84 39 92 a5 d3   ........
      0150:  7c cc 82 5d f3 1c dc ce  3c 8b a5 fc af b2 04 7d   |..]....<.
      0160:  e7 43 94 ca 5a 9d cc 57  1f 61 3f ba b3 f0 27 b4   .C..Z..W.a
      0170:  9e 74 e6 5c d5 12 d0 3e  5b 9b 3f 1e 4f 3a 52 3a   .t.\...>[.
      0180:  fe 58 f7 f3 5c f8 52 ee  48 08 fe 01 13 e2 3d 37   .X..\.R.H.
      0190:  67 3c 2b 53 60 a9 b0 00                           g<+S`...

PS/PES packet (length=292):
  Packet_start_code_prefix: 0x000001
  Stream_id: 3 (0x03)  [= slice_start_code]
    MPEG-2 Slice (incl. sync + id):
      0000:  00 00 01 03 23 9a 9f 9c  17 29 ce 94 a0 bd 33 2e   ....#....)
      0010:  69 8d c8 a5 5a c7 56 0e  61 55 a0 f6 86 6c 10 36   i...Z.V.aU
      0020:  6a b2 31 ba 87 46 19 c7  3d 9d 9e 28 53 52 ef f4   j.1..F..=.
      0030:  c2 fb d0 99 0a 21 34 dd  91 92 56 8d a2 5b a5 42   .....!4...
      0040:  c0 3f 8d 53 be 9c e8 0e  69 93 d5 aa de fa 11 13   .?.S....i.
      0050:  8c 91 ae e4 8a fd 1a 8a  d5 bc 9e 8d 33 8a ad c4   ........
      0060:  48 29 a5 bb ca e7 6c ff  c4 d0 1b 64 00 4e 58 71   H)....l...
      0070:  fe 4f 17 73 30 01 e0 06  b8 68 14 2d 03 f3 c4 31   .O.s0....h
      0080:  a0 37 26 01 44 ba f9 33  0d 15 e4 ba 19 45 36 0b   .7&.D..3..
      0090:  fe 99 85 65 34 8e c6 49  7b 8c 51 33 ec e7 9d 63   ...e4..I{.
      00a0:  72 ff f9 b3 98 38 ff eb  85 79 64 ea 80 ed 37 37   r....8...y
      00b0:  c6 15 7b 19 cf df b0 10  79 22 54 c2 ae b5 19 73   ..{.....y"
      00c0:  31 c7 8b 93 82 57 8e 67  00 8f 10 55 6e ff 3f fe   1....W.g..
      00d0:  52 d2 b3 d8 c3 33 87 9b  66 5e d4 dc dc df 33 26   R....3..f^
      00e0:  47 3f b2 cc 59 16 4e 0e  0f 55 b1 d4 e2 9c 4d db   G?..Y.N..U
      00f0:  aa e6 04 3f 38 d2 3e 69  83 a0 3d 92 b3 03 84 1f   ...?8.>i..
      0100:  a4 d1 05 50 71 bc 36 a9  ec c8 aa c9 62 33 59 5b   ...Pq.6...
      0110:  2a d7 23 31 bc 3d 4b 18  a1 6c aa b1 a1 f5 b2 18   *.#1.=K..l
      0120:  d6 c3 36 00                                       ..6.
```

C. PACKETIZED ELEMENTARY STREAM AUDIO DIFF

Side-by-side diff of two audio PES from different regions. A ‘|” indicates a conflicting line, ‘<” a line only present in the left listing and a ‘>” correspondingly for the right listing.

```
TS sub-decoding (15 packet(s) stored for PID 0x029e):                                <
==========================================================                           <
TS contains PES/PS stream...                                                         <
PS/PES packet (length=2702):                                                         <
    Packet_start_code_prefix: 0x000001                                               <
    Stream_id: 192 (0xc0)  [= ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]       <
    PES_packet_length: 2696 (0x0a88)                                                 <
        reserved1: 2 (0x02)                                                          <
        PES_scrambling_control: 0 (0x00)  [= not scrambled]                          <
        PES_priority: 0 (0x00)                                                       <
        data_alignment_indicator: 1 (0x01)                                           <
        copyright: 0 (0x00)                                                          <
        original_or_copy: 0 (0x00)                                                   <
        PTS_DTS_flags: 2 (0x02)                                                      <
        ES_rate_flag: 0 (0x00)                                                       <
        additional_copy_info_flag: 0 (0x00)                                          <
        PES_CRC_flag: 0 (0x00)                                                       <
        PES_extension_flag: 0 (0x00)                                                 <
        PES_header_data_length: 5 (0x05)                                             <
        PTS:                                                                         <
           Fixed: 2 (0x02)                                                           <
           PTS:                                                                      <
              bit[32..30]: 1 (0x01)                                                  <
              marker_bit: 1 (0x01)                                                   <
              bit[29..15]: 14125 (0x372d)                                            <
              marker_bit: 1 (0x01)                                                   <
              bit[14..0]: 27410 (0x6b12)                                             <
              marker_bit: 1 (0x01)                                                   <
               ==> PTS: 1536617234 (0x5b96eb12)  [= 90 kHz-Timestamp: 4:44:33.524 <
        stuffing bytes:                                                              <
            ff                                                .                      <
        PES_packet_data_bytes:                                                       <
            fc b4 00 5e c7 66 55 55  66 66 66 66 66 22 22 33   ...^.fUUfffff""3
            6d 22 52 48 92 40 00 00  00 00 00 aa 5a aa aa aa   m"RH.@......Z...
            aa aa a0 ff 30 c4 92 4d  94 d9 69 a6 9a 6d b6 18   ....0..M..i..m..
            79 e9 a6 9a 69 65 8e 39  65 9e 79 65 9e 39 a7 8e   y...ie.9e.ye.9..
            6a a4 aa 4a e4 ae 48 1b  33 c0 d9 9e 42 a3 90 a8   j..J..H.3...B...
            e9 e5 da 79 75 f6 23 ec  4b 5c 36 b8 4f 63 1e c6   ...yu.#.K\6.Oc..
            48 de 91 bc 16 70 2c e3  26 5f 3e 93 49 a1 b0 d9   H....p,.&_>.I...
            12 1e f8 d1 8a 14 18 c2  17 13 d4 38 9e a1 e7 6c   ...........8...l
            79 db 2a 7e 5a 9f 96 68  c4 d1 83 e7 27 ce 55 04   y.*~Z..h....'.U.
            aa 09 bb 19 76 31 ac 63  58 c4 a9 52 24 56 2b 2d   ....v1.cX..R$V+-
            76 b8 70 da d7 cf 9f bf  5a da d5 2e ce a9 76 74   v.p.....Z.....vt
            fb d1 3e f4 65 96 39 65  8e 51 74 a2 e6 2a 8c 55   ..>.e.9e.Qt..*.U
            33 e8 67 d0 8b df 17 be  50 7c a0 f1 c3 a1 c3 71   3.g.....P|.....q
            b8 db ad d8 91 12 98 70  c5 8b 31 88 47 d0 1d 3e   .......p..1.G..>
            80 ea 86 22 a1 88 a2 92  18 a4 86 07 ac 0f 59 35   ..."..........Y5
            92 6b 0c c7 19 8e 6b a0  d7 42 4e 74 9c e4 a9 7a   .k....k..BNt...z
            f5 72 b9 64 b2 57 af 5a  d7 cf 92 a5 d6 9a d6 f0   .r.d.W.Z........
            b4 37 85 a2 0a 46 82 91  9f 8b 67 e2 d9 9c 5b 38   .7...F....g...[8
            ba 17 94 2f 1b 12 b6 25  7d 5c fa ba 64 bc c9 78   .../...%}\..d..x
            50 9e bd b1 d8 e4 92 44  68 ce 7b b7 66 cd 10 8e   P.......Dh.{.f...
            7d 6d a4 eb 6d 26 67 e6  99 f9 9a 91 96 a4 65 b0   }m..m&g.......e.
            83 61 0b 29 56 52 8b 41  16 82 03 ea 07 d5 13 7a   .a.)VR.A.......z
            26 f7 ef de 3c 70 b8 63  51 a7 2e 5c e7 cf 9f 3e   &...<p.cQ..\...>
            6b 5a d5 8a 5e ac 52 f6  55 33 95 4c e0 8e 38 23   kZ..^.R.U3.L..8#
            8d ea 7b d4 f7 ca 2f 94  69 73 d2 e7 73 9c e7 39   ..{.../.is..s..9
            ae 8b 5d 16 ed de bd 29  14 8c 66 37 cf 94 a8 70   ..]....)..f7...p
            df 3e 6b 5a d4 4e 9e a2  74 f5 79 9e 5e 67 9e 7d   .>kZ.N..t.y.^g.}
            c7 9f 72 0e 44 1c 85 b6  4b 6c 8f 07 9e 0f 7b 9c   ..r.D...Kl....{.
            f7 3a 11 54 22 a7 cf a9  d3 76 3b 22 91 4a 74 da   .:.T"....v;".Jt.
            da f5 df 3e 6b 5a d7 10  22 b8 81 16 28 e1 8a 38   ...>kZ.."...(..8
            4e 9d 13 a7 46 a3 dd 47  b5 ba 2b 74 5c b7 b9 6f   N...F..G..+t\..o
            74 5a e8 b6 ac 6d 58 d7  4e a9 52 48 a4 63 31 99   tZ...mX.N.RH.c1.
            12 14 a1 a3 5c b9 21 1a  db 9e a0 5c f5 01 3c 28   ....\.!....\..<(
            4f 0a 04 c8 b1 32 2c 5c  28 b8 56 9e 8d 3d 2c e6   O....2,\(.V..=,.
            d9 cd 73 a2 e7 46 c6 8d  8d 1c 38 5e bd 49 a4 c0   ..s..F....8^.I..
            60 3b 97 06 35 0a 20 c1  6b 5c e8 0d ce c0 6e 75   `;..5. .k\....nu
            87 58 61 d6 25 4d 69 53  5a 6d 9c db 38 3e 50 7c   .Xa.%MiSZm..8>P|
            a2 a8 c5 51 8b 9d 17 3a  2e 9c 5d 38 50 9f 3e 8a   ...Q...:..]8P.>.

                                                                                     fc b4 00 5e c7 66 55 55  66 66 66 66 66 22 22 33   ...^.fUUfffff""3
                                                                                     6d 22 52 48 92 40 00 00  00 00 00 aa 5a aa aa aa   m"RH.@......Z...
                                                                                     aa aa a0 ff 30 c4 92 4d  94 d9 69 a6 9a 6d b6 18   ....0..M..i..m..
                                                                                     79 e9 a6 9a 69 65 8e 39  65 9e 79 65 9e 39 a7 8e   y...ie.9e.ye.9..
                                                                                     6a a4 aa 4a e4 ae 48 1b  33 c0 d9 9e 42 a3 90 a8   j..J..H.3...B...
                                                                                     e9 e5 da 79 75 f6 23 ec  4b 5c 36 b8 4f 63 1e c6   ...yu.#.K\6.Oc..
                                                                                     48 de 91 bc 16 70 2c e3  26 5f 3e 93 49 a1 b0 d9   H....p,.&_>.I...
                                                                                     12 1e f8 d1 8a 14 18 c2  17 13 d4 38 9e a1 e7 6c   ...........8...l
                                                                                     79 db 2a 7e 5a 9f 96 68  c4 d1 83 e7 27 ce 55 04   y.*~Z..h....'.U.
                                                                                     aa 09 bb 19 76 31 ac 63  58 c4 a9 52 24 56 2b 2d   ....v1.cX..R$V+-
                                                                                     76 b8 70 da d7 cf 9f bf  5a da d5 2e ce a9 76 74   v.p.....Z.....vt
                                                                                     fb d1 3e f4 65 96 39 65  8e 51 74 a2 e6 2a 8c 55   ..>.e.9e.Qt..*.U
                                                                                     33 e8 67 d0 8b df 17 be  50 7c a0 f1 c3 a1 c3 71   3.g.....P|.....q
                                                                                     b8 db ad d8 91 12 98 70  c5 8b 31 88 47 d0 1d 3e   .......p..1.G..>
                                                                                     80 ea 86 22 a1 88 a2 92  18 a4 86 07 ac 0f 59 35   ..."..........Y5
                                                                                     92 6b 0c c7 19 8e 6b a0  d7 42 4e 74 9c e4 a9 7a   .k....k..BNt...z
                                                                                     f5 72 b9 64 b2 57 af 5a  d7 cf 92 a5 d6 9a d6 f0   .r.d.W.Z........
                                                                                     b4 37 85 a2 0a 46 82 91  9f 8b 67 e2 d9 9c 5b 38   .7...F....g...[8
                                                                                     ba 17 94 2f 1b 12 b6 25  7d 5c fa ba 64 bc c9 78   .../...%}\..d..x
                                                                                     50 9e bd b1 d8 e4 92 44  68 ce 7b b7 66 cd 10 8e   P.......Dh.{.f...
                                                                                     7d 6d a4 eb 6d 26 67 e6  99 f9 9a 91 96 a4 65 b0   }m..m&g.......e.
                                                                                     83 61 0b 29 56 52 8b 41  16 82 03 ea 07 d5 13 7a   .a.)VR.A.......z
                                                                                     26 f7 ef de 3c 70 b8 63  51 a7 2e 5c e7 cf 9f 3e   &...<p.cQ..\...>
                                                                                     6b 5a d5 8a 5e ac 52 f6  55 33 95 4c e0 8e 38 23   kZ..^.R.U3.L..8#
                                                                                     8d ea 7b d4 f7 ca 2f 94  69 73 d2 e7 73 9c e7 39   ..{.../.is..s..9
                                                                                     ae 8b 5d 16 ed de bd 29  14 8c 66 37 cf 94 a8 70   ..]....)..f7...p
                                                                                     df 3e 6b 5a d4 4e 9e a2  74 f5 79 9e 5e 67 9e 7d   .>kZ.N..t.y.^g.}
                                                                                     c7 9f 72 0e 44 1c 85 b6  4b 6c 8f 07 9e 0f 7b 9c   ..r.D...Kl....{.
                                                                                     f7 3a 11 54 22 a7 cf a9  d3 76 3b 22 91 4a 74 da   .:.T"....v;".Jt.
                                                                                     da f5 df 3e 6b 5a d7 10  22 b8 81 16 28 e1 8a 38   ...>kZ.."...(..8
                                                                                     4e 9d 13 a7 46 a3 dd 47  b5 ba 2b 74 5c b9 6f      N...F..G..+t\.o
                                                                                     74 5a e8 b6 ac 6d 58 d7  4e a9 52 48 a4 63 31 99   tZ...mX.N.RH.c1.
                                                                                     12 14 a1 a3 5c b9 21 1a  db 9e a0 5c f5 01 3c 28   ....\.!....\..<(
                                                                                     4f 0a 04 c8 b1 32 2c 5c  28 b8 56 9e 8d 3d 2c e6   O....2,\(.V..=,.
                                                                                     d9 cd 73 a2 e7 46 c6 8d  8d 1c 38 5e bd 49 a4 c0   ..s..F....8^.I..
                                                                                     60 3b 97 06 35 0a 20 c1  6b 5c e8 0d ce c0 6e 75   `;..5. .k\....nu
                                                                                     87 58 61 d6 25 4d 69 53  5a 6d 9c db 38 3e 50 7c   .Xa.%MiSZm..8>P|
                                                                                     a2 a8 c5 51 8b 9d 17 3a  2e 9c 5d 38 50 9f 3e 8a   ...Q...:..]8P.>.
```

```
45 24 92 4b 97 2d 64 a9  5f 3e 6b 5a d6 ca 1a b6   E$.K.-d._>kZ....
50 d6 58 22 96 08 9d 7d  b7 5f 6d b0 6b 60 d8 be   P.X"...}._m.k`..
11 7c 1a c6 35 8c 8c 1d  18 39 ac 73 58 e7 cf ab   .|..5....9.sX...
d7 4d 26 93 69 b7 cf 9a  d8 91 1e bd 6b 5a dd ff   .M&.i.......kZ..
fc b4 00 06 9c 66 55 55  66 66 66 66 44 44 22 33   .....fUUffffDD"3
6d 22 52 48 92 40 00 00  00 00 00 aa af 0a aa aa   m"RH.@..........
aa aa a0 00 38 e6 18 6d  b7 1e 71 e7 21 79 c8 5e   ....8..m..q.!y.^
7d f8 61 8e 39 65 a2 89  e7 a2 89 a6 a2 89 e7 9e   }.a.9e..........
b9 a7 ae 6a 6e 9e 9b a7  a6 ea 29 ba 86 cc 61 36   ...jn.....)...a6
63 09 dd 54 77 55 2d b2  ab 6c a9 ed 5b da b7 47   c..TwU-..l..[..G
8e 8f 30 92 e1 25 c5 9b  8b 35 90 64 2c 4b 11 e3   ..0..%...5.d,K..
cc 96 49 da ed b3 64 a5  48 91 02 0d 69 8c 7f 3f   ..I...d.H...i..?
eb f9 ff 50 4e 64 13 98  46 09 11 82 66 2a cc 55   ...PNd..F...f*.U
7c 20 f8 41 4e 82 9d 00  5d 80 bb 08 32 0c 73 1c   | .AN...]...2.s.
d2 24 52 a9 53 09 87 af  73 98 b1 58 b1 73 94 af   .$R.S...s..X.s..
b2 15 fd 90 af 37 e2 cd  f8 a9 ba 4a 6e 91 95 5b   .....7.....Jn..[
2a b7 c5 cf 8b 93 35 a6  6b 7b e0 f7 c1 5c 57 15   *.....5.k{...\W.
c5 72 14 2b 55 a8 20 10  74 eb 39 7c f9 f3 e6 b5   .r.+U. .t.9|....
ad 30 75 69 83 ab 4d 5f  63 57 da 17 01 85 c0 6c   .0ui..M_cW.....l
a3 d9 47 43 90 87 21 cf  43 9e 8a 51 34 a2 65 f9   ..GC..!.C..Q4.e.
7d 6d 5b 6e 5c 09 04 9b  2d 97 cf 9c e7 cf 9f 3e   }m[n\....-......>
8c 5a d7 6f 1a bb 78 d5  75 eb 5d 7a ea 6a 8a 9a   .Z.o..x.u.]z.j..
a1 4a ba 95 72 14 64 28  d7 08 ae 11 82 dd 05 b9   .J..r.d(........
62 58 9d 17 45 22 48 ac  55 d2 e9 af 5d 8c 87 0e   bX..E"H.U...]...
04 0b dd 4a 9b 21 b4 d9  0d a0 a6 08 29 81 74 97   ...J.!......).t.
5d 25 e4 a6 c9 4d cb 01  96 01 65 7a ca f5 b6 cb   ]%...M....ez....
6d 9c e7 39 86 61 a7 4e  11 08 80 40 27 ef ed 67   m..9.a.N...@'..g
cf a1 c3 5a ce 79 d8 f3  ce c7 9f 61 1e d8 47 a7   ...Z.y.....a..G.
ca 69 f2 98 f2 21 e4 4b  ca d7 95 8e a7 1d 4e 9b   .i...!.K......N.
ed 37 d8 ca 32 92 64 99  0a 18 dc 6c a8 54 4c 99   .7..2.d....l.TL.
4a 72 e6 85 06 b4 21 a2  f5 45 17 aa 29 91 5a 64   Jr....!..E..).Zd
56 33 17 8c c5 ea e2 d5  c5 41 6c 82 da 30 9c 61   V3.......Al..0.a
35 c0 cb 81 a5 59 56 69  9a 69 52 26 13 2b 55 a0   5....YVi.iR&.+U.
e1 eb 5e 1c 1c b9 18 de  f3 8a 14 9c 50 a6 68 1b   ..^.........P.h.
9a 06 d5 41 f5 50 7d d5  6b aa d3 3b 86 77 25 14   ...A.P}.k..;.w%.
ca 29 4e 22 9c 45 80 60  23 38 cd 02 03 09 84 20   .)N".E.`#8.....
10 7c f9 8c 3e 7c 30 66  b5 ad 4e fe 62 77 f3 1f   .|..>|0f..N.bw..
9a 37 e6 8d d3 8c 74 e3  31 ac 63 58 6c a6 d9 4d   .7....t.1.cXl..M
cf 2b 9e 50 dd e1 bb e6  49 91 d8 76 1f bf 53 29   .+.P....I..v..S)
9c ae 57 ef d8 c4 89 11  a3 73 8c 68 12 25 40 91   ..W......s.h.%@.
2a 15 d4 85 75 27 0d 19  c3 44 52 38 a4 71 b5 43   *...u'...DR8.q.C
6a b7 84 ef 09 b6 57 6c  ad 77 5d d6 25 8a dd b6   j.....Wl.w].%...
9b 4d 68 b4 7a f5 ad 95  29 02 07 bd 8c 9d 72 cc   .Mh.z...).....r.
eb 96 61 75 18 5d 45 4c  8c 53 23 26 e8 cd d1 d4   ..au.]EL.S#&....
51 a8 a1 ae 83 5d 03 3e  26 7c 5d 87 61 b2 6c a2   Q....].>&|].a.l.
44 a9 d4 e5 52 a7 af 5e  f7 af 5f 3e 6b 5a de ff   D...R..^.._>kZ..  |
                                                                    >
                                                                    >
                                                                    >     TS sub-decoding (15 packet(s) stored for PID 0x029e):
                                                                    >     ====================================================
                                                                    >     TS contains PES/PS stream...
                                                                    >     PS/PES packet (length=2702):
                                                                    >         Packet_start_code_prefix: 0x000001
                                                                    >         Stream_id: 192 (0xc0)  [= ISO/IEC 13818-3 or ISO/IEC 11172-3 audio stream]
                                                                    >         PES_packet_length: 2696 (0x0a88)
                                                                    >             reserved1: 2 (0x02)
                                                                    >             PES_scrambling_control: 0 (0x00)  [= not scrambled]
                                                                    >             PES_priority: 0 (0x00)
                                                                    >             data_alignment_indicator: 1 (0x01)
                                                                    >             copyright: 0 (0x00)
                                                                    >             original_or_copy: 0 (0x00)
                                                                    >             PTS_DTS_flags: 2 (0x02)
                                                                    >             ES_rate_flag: 0 (0x00)
                                                                    >             additional_copy_info_flag: 0 (0x00)
                                                                    >             PES_CRC_flag: 0 (0x00)
                                                                    >             PES_extension_flag: 0 (0x00)
                                                                    >             PES_header_data_length: 5 (0x05)
                                                                    >             PTS:
                                                                    >                Fixed: 2 (0x02)
                                                                    >                PTS:
                                                                    >                   bit[32..30]: 1 (0x01)
                                                                    >                   marker_bit: 1 (0x01)
                                                                    >                   bit[29..15]: 598 (0x0256)
                                                                    >                   marker_bit: 1 (0x01)
                                                                    >                   bit[14..0]: 27294 (0x6a9e)
                                                                    >                   marker_bit: 1 (0x01)
                                                                    >                    ==> PTS: 1093364382 (0x412b6a9e)  [= 90 kHz-Timestamp: 3:22:28.493
                                                                    >             stuffing bytes:
                                                                    >                 ff                                                .
                                                                    >             PES_packet_data_bytes:
fc b4 00 3b e9 66 55 55  66 66 66 55 55 44 33 22   ...;.fUUfffUUD3"      fc b4 00 3b e9 66 55 55  66 66 66 55 55 44 33 22   ...;.fUUfffUUD3"
6d 24 92 24 92 40 00 00  00 00 00 a5 5a f0 aa fa   m$.$.@......Z...      6d 24 92 24 92 40 00 00  00 00 00 a5 5a f0 aa fa   m$.$.@......Z...
af 0a 0a aa 34 d6 15 61  56 d9 6d 97 1c 81 c8 1c   ....4..aV.m.....      af 0a 0a aa 34 d6 15 61  56 d9 6d 97 1c 81 c8 1c   ....4..aV.m.....
76 17 9d 85 e8 a2 8a 29  e3 9e 39 a6 9e 7a 24 a2   v.......)..9..z$.      76 17 9d 85 e8 a2 8a 29  e3 9e 39 a6 9e 7a 24 a2   v.......)..9..z$.
4a 66 a6 99 a9 a2 8a 25  a6 89 69 aa aa 69 a6 98   Jf.....%..i..i..      4a 66 a6 99 a9 a2 8a 25  a6 89 69 aa aa 69 a6 98   Jf.....%..i..i..
8e 56 c4 72 b4 05 24 01  49 24 69 f9 1a 7d 4f 8a   .V.r..$.I$i..}O.      8e 56 c4 72 b4 05 24 01  49 24 69 f9 1a 7d 4f 8a   .V.r..$.I$i..}O.
9f 15 5e 6a bc da e8 b5  d1 8a 58 a5 73 87 38 2d   ..^j......X.s.8-      9f 15 5e 6a bc da e8 b5  d1 8a 58 a5 73 87 38 2d   ..^j......X.s.8-
cb 76 43 20 38 76 43 20  81 04 c9 b1 62 21 1a db   .vC 8vC ....b!..      cb 76 43 20 38 76 43 20  81 04 c9 b1 62 21 1a db   .vC 8vC ....b!..
59 02 66 28 13 31 a5 a1  69 68 57 a4 2d e9 0b 44   Y.f(.1..ihW.-..D      59 02 66 28 13 31 a5 a1  69 68 57 a4 2d e9 0b 44   Y.f(.1..ihW.-..D
28 88 52 9a 95 35 2a b9  f5 73 ee 5e e5 e9 a2 9a   (.R..5*..s.^....      28 88 52 9a 95 35 2a b9  f5 73 ee 5e e5 e9 a2 9a   (.R..5*..s.^....
16 35 8e 95 4a 52 a6 c7  63 54 a9 83 07 cf ad 65   .5..JR..cT.....e      16 35 8e 95 4a 52 a6 c7  63 54 a9 83 07 cf ad 65   .5..JR..cT.....e
2a b5 85 2a 84 29 54 09  76 72 5d 9f 99 e8 e6 7a   *..*.)T.vr]....z      2a b5 85 2a 84 29 54 09  76 72 5d 9f 99 e8 e6 7a   *..*.)T.vr]....z
0e b8 9d 71 6c 74 d8 e9  a2 73 44 ea 51 a5 15 a2   ...qlt...sD.Q...      0e b8 9d 71 6c 74 d8 e9  a2 73 44 ea 51 a5 15 a2   ...qlt...sD.Q...
5a 29 aa 6a 35 1a 8f 9f  88 44 3a f5 50 a2 3c 7b   Z).j5....D:.P.<{      5a 29 aa 6a 35 1a 8f 9f  88 44 3a f5 50 a2 3c 7b   Z).j5....D:.P.<{
```